STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKKA JA LIIKENNE

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

PRIMAVISTAN JAVA-KOODIN OBFUSKOINTIPROSESSI

Työn tekijä: Vesa Herranen
Työn valvoja: Simo Silander
Työn ohjaaja: Mikko Koitto

Työ hyväksytty: __. __. 2007

Simo Silander
lehtori

ALKULAUSE

Tämä insinöörityö tehtiin kesän 2007 aikana Helsingissä Gordion-talousohjaus Oy:lle. Haluan kiittää koko Gordionin henkilökuntaa sekä etenkin toimitusjohtaja Paavo Tähtistä ja kehityspäällikkö Mikko Koittoa mahdollisuudesta tämän työn tekemiseen sekä koko tuotekehitystiimin työntekijöitä avusta prosessin aikana.

Espoossa 22.11.2007

Vesa Herranen

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Vesa Herranen

Työn nimi: Primavistan Java-koodin obfuskointiprosessi

Päivämäärä: 22.11.2007 Sivumäärä: 44 s. + 6 liitettä

Koulutusohjelma: Tietotekniikka Suuntautumisvaihtoehto: Ohjelmistotekniikka

Työn valvoja: lehtori Simo Silander

Työn ohjaaja: kehityspäällikkö Mikko Koitto

Tässä insinöörityössä suunniteltiin ja toteutettiin Gordion-talousohjaus Oy:n Primavista-ohjelmiston koodin suojaaminen. Yrityksellä oli jo pitkään ollut tarve Java-kielisen ohjelmakoodinsa suojaamiselle, etenkin Primavistan käytön tullessa laajentumaan useisiin eri maihin. Koodin suojaamisella hankaloitetaan mahdollista takaisinkääntämistä, jossa valmiin ohjelman ajettavat tiedostot käännetään takaisin niiden lähdekoodiksi. Tällöin lähdekoodia pystytään tarkastelemaan tai jopa muokkaamaan. Suomen tekijänoikeuslain mukaan takaisinkääntäminen ei ole sallittua.

Työ aloitettiin tutkimalla erilaisia Java-koodin suojausmenetelmiä. Primavistan koodin suojausmenetelmäksi valittiin obfuskointi, joka on suosituin Java-koodin suojausmenetelmä. Erilaisia tekniikoita käyttäen tämä menetelmä sotkee Javan tavukoodia tehden takaisinkäännetystä koodista erittäin vaikeasti ymmärrettävää.

Työssä vertailtiin nykyisiä obfuskointityökaluja itse tehdyn testisovelluksen avulla. Vertailun parhaaksi työkaluksi osoittautui kaupallinen tuote Zelix KlassMaster. Ohjelma todettiin toimivaksi myös Primavistan suhteen, minkä jälkeen se päätettiin hankkia yritykselle.

Koko Primavistan koodin obfuskointi toteutettiin Zelix KlassMasteria ja sen useita eri obfuskointitekniikoita käyttäen. Tämän jälkeen obfuskointi liitettiin Primavistan Ant-buildiin kiinteäksi osaksi koodin rakennusprosessia. Koska obfuskointi tekee huomattavia muutoksia tavukoodiin, obfuskoidun Primavistan toiminta oli testattava huolellisesti. Testauksessa huomattiin pieniä virhetilanteita, jotka korjaamalla ohjelman toiminta lopulta todettiin normaaliksi. Viimeisessä työvaiheessa kaikki obfuskoinnissa tarvittavat tiedostot siirrettiin versionhallintaan kaikkien yrityksen kehitystiimin jäsenten käytettäviksi.

Työn tuloksena saatiin onnistuneesti obfuskoiduksi koko Primavistan koodi. Obfuskointiprosessista kehitettiin pysyvä käytäntö, jonka mukaisesti obfuskointi tehdään aina Primavistan rakennusvaiheessa. Jatkossa kaikki asiakkaille toimitettavat Primavistan versiot tulevat olemaan obfuskoituja sovelluksia.

Avainsanat: takaisinkääntäminen, koodin suojaaminen, Java, obfuskointi, Zelix KlassMaster

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Vesa Herranen

Title: The Obfuscation Process of Primavista's Java Code

Date: 22 November 2007 Number of pages: 44

Department: Faculty of Technology Study Programme: Software Engineering

Instructor: Simo Silander, Senior Lecturer

Supervisor: Mikko Koitto, Chief of Development

The purpose of this final year project was to design and implement a code protection for the Primavista software in use at Gordion Business Consulting Ltd. The company has been aware for some time that it needs to protect its Java code, and even more so now that the use of Primavista will be expanded to several other countries. The aim of code protection is to hamper the possible decompilation whereby the program's run-time files are converted back to their source code. Decompilation means that the source code can be observed or even revised. According to the Finnish copyright law decompiling is prohibited.

This study is based on examining different protection methods for Java code. Obfuscation, the most popular code protection method for Java, was selected for protecting Primavista. By using different techniques this method disturbs Java byte code and makes decompiled code very hard to understand.

Current obfuscation tools were compared in this work with the help of a self-made testing application. The best tool in the comparison turned out to be a commercial product called Zelix KlassMaster. The software was found to be working also with Primavista. A decision was made to purchase this tool for the company.

The whole code obfuscation process for Primavista was carried out by using Zelix KlassMaster and its several different obfuscation techniques. After this, the obfuscation was integrated into Primavista's Ant-build as a solid part of the code building process. As obfuscation makes major modifications to the byte code, the obfuscated Primavista functionality had to be thoroughly tested. Some minor malfunctions were found during the testing. After these errors were fixed, the program's functionality was found to be normal. Finally, all obfuscation related files were transferred to version control to make them available for use for all members of the company's development team.

Based on the findings of this work, it was possible to successfully perform the obfuscation for the entire code of Primavista. In addition, the obfuscation process was implemented as a permanent procedure to always take place during the building phase of Primavista. Thus, in the future, all versions of Primavista delivered to customers will be obfuscated applications.

Keywords: decompiling, code protecting, Java, obfuscation, Zelix KlassMaster

SISÄLLYS

ALKULAUSE

TIIVISTELMÄ

ABSTRACT

SISÄLLYS

1 JOHDANTO

Tässä insinöörityössä kerrotaan Gordion-talousohjaus Oy:n Java-kielellä ohjelmoidun Primavista-ohjelmiston koko koodin suojausprosessin suunnittelusta ja toteutuksesta. Gordionilla on jo pidempään ollut tarve ohjelmakoodinsa suojaamiselle, ja asia tuli entistäkin ajankohtaisemmaksi kesällä 2007 tapahtuneen Primavistan yritysmyynnin vuoksi. Kaupan myötä ohjelmiston käyttö tulevaisuudessa laajentuu useisiin eri maihin, myös Euroopan ulkopuolelle.

Menetelmänä tässä prosessissa on käytetty Java-kielen suosituinta koodin suojausmenetelmää, obfuskointia. Erilaisia tekniikoita käyttäen tämä menetelmä sotkee koodia, ja siten tekee siitä takaisinkääntämisen jälkeen erittäin hankalasti ymmärrettävää. Tähän prosessiin kuului olemassa olevien obfuskointityökalujen vertailu, parhaan työkalun valinta ja sitä käyttäen Primavistan ohjelmakoodin obfuskoiminen sekä obfuskoidun ohjelman toiminnan testaaminen. Obfuskointityökaluksi valikoitui kaupallinen Zelix KlassMaster -ohjelma.

Ohjelmakoodin takaisinkääntämisellä (*reverse engineering* tai *decompilation*) tarkoitetaan toimenpidettä, jossa valmiin ohjelman ajettavat tiedostot käännetään takaisin niiden lähdekoodiksi [1]. Tällöin lähdekoodia pystytään tarkastelemaan tai jopa muokkaamaan. Takaisinkääntäminen on suuri haitta etenkin kaupallisten ohjelmistojen kannalta. Suomen tekijänoikeuslain mukaan takaisinkääntäminen ei ole sallittua [2; 3].

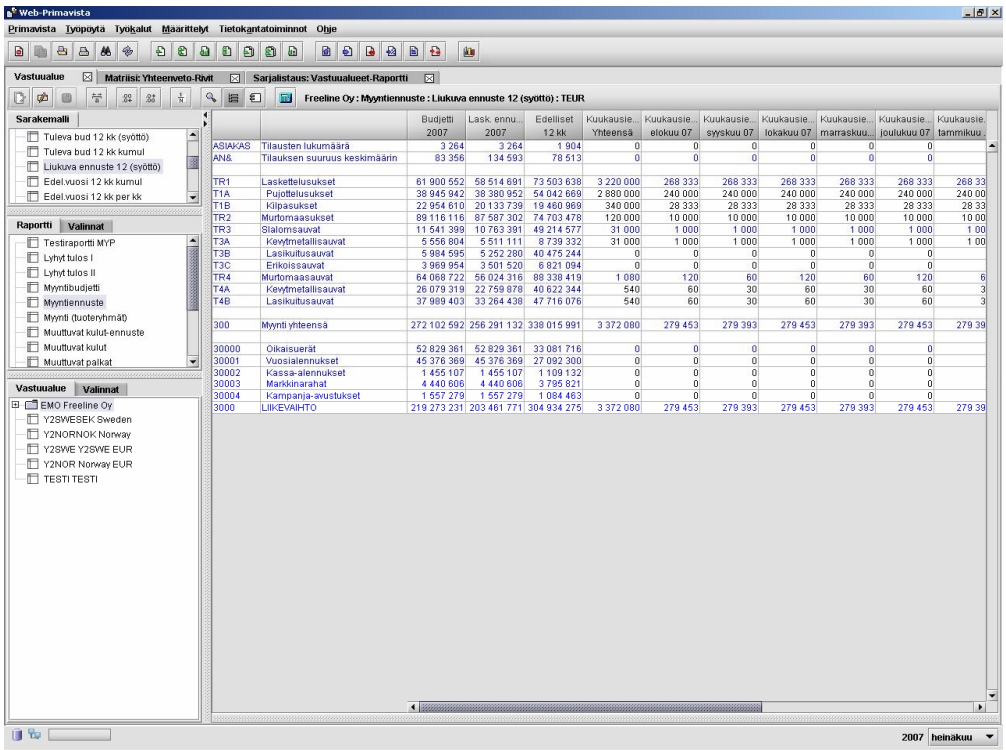
Käytetystä ohjelmointikielestä riippuen takaisinkääntäminen ja takaisinkäännetyn koodin ymmärtäminen voivat viedä huomattavasti aikaa. Asiansa osaava ja tarpeeksi päättäväinen takaisinkääntäjä pystyy nämä toimenpiteet kuitenkin aina tekemään. Tästä syystä ohjelmistoista puhuttaessa mitään liike-ideoita ei ole mahdollista täysin salata eikä estää ulkopuolisten tahojen pääsyä muokkaamaan koodia. Tämä on merkittävä puute esimerkiksi ohjelmistojen lisenssienhallinnan kannalta. Takaisinkääntämisen hankaloittamiseksi on kuitenkin olemassa erilaisia keinoja suojata ja salata ohjelmakoodia.

2 GORDION-TALOUSOHJAUS OY

Gordion-talousohjaus Oy on vuonna 1988 perustettu pieni, 10-henkinen johtamisjärjestelmien ja talousohjauksen asiantuntija- ja tietotekniikkayritys. Gordion valmistaa Primavista-ohjelmistoperhettä, myy ja toimittaa siihen perustuvia asiakaskohtaisia järjestelmäratkaisuja ja tarjoaa asiakkaillensa tukipalveluita sen käytössä. Yrityksen keskeisimpänä palveluna on auttaa asiakkaitaan hyödyntämään tehokkaasti nykyisten tietojärjestelmiensä tuottamaa informaatiota.

2.1 Primavista-tuoteperhe

Primavista-tuoteperhe on SQL-rajapintaisiin relaatiotietokantoihin perustuva tietovarastojärjestelmä, jota käytetään liiketoiminnan johtamisen työkaluna. Kyseessä on hybridituote, jonka tehokäyttäjäversiot on tarkoitettu talous- ja laskenta-asioiden ammattilaisille. Ohjelmistoperheen Web-Primavista-moduulit (kuva 1) puolestaan on tarkoitettu käytön hajautukseen läpi koko organisaation. Tuoteperhe tarjoaa apuvälineistön asiakasyrityksen kokonaisuuden hallintatarpeisiin ja toimintaa toteuttavien yksiköiden omiin tarpeisiin liiketoiminnan suunnittelussa, ohjauksessa, analysoinnissa ja seurannassa.



Kuva 1. Web-Primavistan asiakassovelluksen raporttinäkymä testitietokannalla

Primavista täydentää yrityksen perusjärjestelmiä ja esimerkiksi ERP-järjestelmiä tarjoamalla niiden tueksi työkalun, jolla voidaan automatisoida ne toimenpiteet, jotka yleensä tehdään paljon työtä vaativilla taulukkolaskinmalleilla. Primavistan käyttöalueita ovat muun muassa erilaiset seuranta-raportoinnit, tiedon analysointi, budjetointi, ennustaminen, vuosisuunnittelu sekä konsernilaskenta.

Tässä työssä käsiteltävä Web-Primavista on ohjelmoitu kokonaan Java-kielellä. Web-Primavistan keskeinen idea on, että useat asiakassovellukset voivat olla yhtä aikaa yhteydessä palvelinsovellukseen. Graafista, Javan Swing-kirjastolla toteutettua asiakassovellusta voi käyttää lähiverkon ohessa helposti esimerkiksi intra- tai internetin kautta Javan Web Start -tekniikan avulla. Asiakas- ja palvelinsovellus kommunikoivat keskenään Javan RMI-hajautustekniikkaa käyttäen ja palvelinsovellus on yhteydessä asiakasyrityksen tietokantoihin Javan JDBC-tekniikan avulla (kuva 2).

Kuva 2. Web-Primavistan arkkitehtuurikaavio

2.2 Tuotekehityksessä käytettävät työkalut

Web-Primavista on ohjelmoitu Javan JDK 1.5 -kehityspakettia käyttäen. Kehitysympäristönä käytetään ilmaisen lähdekoodin ohjelmaa Eclipseä, joka on yksi maailman suosituimmista ohjelmointityökaluista. Eclipsen parhaita puolia – ilmaisuuden ja avoimen lähdekoodin lisäksi – ovat tuki usealle eri ohjelmointikielelle sekä helppo laajennettavuus erilaisia lisäosia käyttämällä.

Varsinaisena Java-koodin rakennus- ja käännöstyökaluna käytetään Eclipsen automaattisesti tukemaa Apache Antia, jonka XML-pohjainen rakenne on hyödyllinen koko ohjelmiston rakennusprosessissa. Antia käytetään muun muassa Java-lähdekoodin kääntämisessä tavukoodimuotoon sekä tavukooditiedostojen pakkaamisessa Javan JAR-paketteihin.

Primavistan lähdekoodin versionhallintaohjelmana käytetään Subversionia, joka on Eclipseen liitetty avoimen lähdekoodin ohjelma. Projektinhallintaohjelmana puolestaan käytetään Tracia, joka Subversionin tavoin on ilmainen

avoimen lähdekoodin ohjelma. Trac tarjoaa web-pohjaisen käyttöliittymän projektin osatehtävien seurannalle.

3 JAVA-KOODIN SUOJAAMINEN

Java on Sun Microsystemsin kehittämä yksi suosituimmista olio-ohjelmointikielistä [4]. Java-sovelluksen teko aloitetaan Java-kielisen lähdekoodin kirjoittamisella. Tämä lähdekoodi käännetään järjestelmäriippumattomaksi tavukoodiksi Java-kääntäjällä, jollainen löytyy sisäänrakennettuna esimerkiksi Eclipsestä. Suoritettavat tavukooditiedostot puolestaan ajetaan Java-ajoympäristöstä (Java Runtime Environment) löytyvällä Java-virtuaalikoneella (Java Virtual Machine). Kääntäminen tavukooditiedostoiksi nopeuttaa sovelluksen suorittamista ja pienentää sen kokoa. [5, s. 10, 13.]

Javan tavoin jotkin muutkin ohjelmointikielet, kuten esimerkisi Microsoftin .NET-arkkitehtuuria käyttävät kielet, käyttävät tavukoodikääntämistä hyödykseen. Tavukooditiedostojen käytöstä poiketen perinteinen tapa on kääntää sovellukset suoraan ajettaviksi konekielisiksi tiedostoiksi. Tällaista menettelyä käyttäviä ohjelmointikieliä ovat esimerkiksi C ja C++.

Koska tavukooditiedostojen on annettava tarkkaa tietoa koodista Java-virtuaalikoneelle, tavukoodit ovat helpommin takaisinkäännettävissä kuin konekieliset tiedostot. Tästä syystä tavukooditiedostot ovat myös erittäin helposti takaisinkäännettävissä. Myös konekielisiä sovelluksia pystyy takaisinkääntämään, mutta se on huomattavasti vaikeampaa kuin tavukooditiedostojen tapauksessa. [6.]

Takaisinkääntämisen estämiseksi on kehitetty erilaisia suojauskeinoja. Javan suosituin tavukoodin suojausmenetelmä on obfuskointi, joka sotkee tavukooditiedostoja ja pienentää niiden kokoa. Muita keinoja ovat esimerkiksi enkryptaus ja Java-koodin kääntäminen konekielelle. Myös omatekoisilla algoritmeilla on mahdollista suojata koodiaan. Tällöin on kuitenkin oltava syvällinen ymmärrys Java-tekniikasta ja osattava ohjelmoida erittäin vaativia algoritmeja. Yleensä omatekoisilla salausmenetelmillä ei saadakaan merkittävää hyötyä aikaan, vaan tavallisesti käytetään valmiita suojausmenetelmiä ja niitä tarjoavia työkaluja.

3.1 Konekielikäännös

Javan konekielikäännöksellä tarkoitetaan toimenpidettä, jossa erillisellä ohjelmalla käännetään Javan tavukooditiedosto konekieliseksi .exe-tiedostoksi [7]. Javan kehityksestä vastaava Sun ei tällaisia sovelluksia ole omaan JDK-pakettiinsa sisällyttänyt. Jotkin koodin suojaamiseen erikoistuneet kaupalliset Javan konekielikääntäjät tarjoavat myös lisätoimintoja, joilla takaisinkääntämistä voidaan vaikeuttaa suojaamalla konekielitiedostoja.

Vaikka konekielikäännöksellä vaikeutettaisiinkin takaisinkääntämistä, sitä käyttämällä myös menetettäisiin Javan tavukooditiedostojen tarjoama järjestelmäriippumattomuus. Tämä johtuu siitä, että konekieli on aina suunniteltu jotain tiettyä järjestelmää varten. Myöskään Javan Web Start -tekniikka ei olisi konekielisten .exe-tiedostojen tapauksessa käytettävissä, sillä Web Start toimii ainoastaan Javan JAR-tiedostojen kanssa. Tästä syystä konekielikäännös ei ole erityisen varteenotettava vaihtoehto Javalle, etenkään Web Start -tekniikkaa käyttävän Primavistan tapauksessa. Javan konekielikäännöstä vastaava hyöty saataisiinkin yksinkertaisesti valitsemalla ohjelmointikieleksi joku konekielikäännöstä käyttävä kieli.

3.2 Enkryptaus

Enkryptauksella tarkoitetaan toimenpidettä, jossa data salataan erilaisia algoritmeja käyttäen ja salaus puretaan tietyn salaisen avaimen avulla [8]. Sun on sisällyttänyt JDK-pakettiinsa useita kryptografisia ominaisuuksia, kuten enkryptauksen [9].

Pelkästään enkryptausta ei voida pitää riittävän toimivana koodin suojauskeinona, sillä se on helposti purettavissa ratkaisemalla vain yhden salaukseen käytettävän algoritmin rakenne. Markkinoilla on kuitenkin kaupallisia ohjelmistoja, jotka tarjoavat koodin suojausmenetelmäksi enkryptauksen ja obfuskoinnin yhdistelmän. Tällainen yhdistelmä olisi toimiessaan pelkkää obfuskointia turvallisempi vaihtoehto.

Enkryptaus ei kuitenkaan ole erityisen käytännöllinen tekniikka, sillä sen toteutus ja ylläpito ovat erittäin hankalia ja aikaa vieviä toimenpiteitä. Enkryptauksessa käytettävän salauksen purkaminen sovelluksen suorituksen aikana myös hidastaa huomattavasti ajonaikaista toimintaa. Lisäksi enkryptaukseen käytettävät salausalgoritmit kasvattavat tavukooditiedostojen kokoa.

Sen käyttö onkin mahdollista ainoastaan niille sovelluksille, joiden käyttönopeus ei ole kriittinen asia, toisin kuin esimerkiksi Primavistan tapauksessa.

3.3 Obfuskointi

Englanninkielinen sana *obfuscation* tarkoittaa jonkin asian muuttamista vaikeammin ymmärrettäväksi. Obfuskointi on keino suojata Java-kielistä lähdekoodia muuttamalla ja sotkemalla Java-virtuaalikoneiden suorittamaa tavukoodia. Obfuskoinnin kehitys on tapahtunut pääasiassa sekä ilmaisten että kaupallisten sovellusten myötä, eikä aiheesta ole olemassa juurikaan tieteellisiä julkaisuja tai kirjallisuutta. Javan suosion kasvun myötä myös obfuskoinnin käyttö on yleistynyt huomattavasti 2000-luvulla.

Itse tehtynä ja suunniteltuna useita eri tekniikoita käyttävä obfuskointi olisi erittäin vaativa toimenpide. Tästä syystä obfuskointiin käytetäänkin lähes aina valmiita työkaluja. Tällaiset työkalut usein sisältävät myös ominaisuuksia, jotka on suunnattu tarkasti takaisinkääntäjäohjelmia vastaan hyödyntämällä niiden tunnettuja heikkouksia ja bugeja.

Obfuskoinnin ideana on muuttaa ja sotkea tavukoodin ulos näkyvää rakennetta siten, ettei itse sovelluksen toimintalogiikka muutu mitenkään. Täten ohjelman toiminta pysyy samanlaisena kuin aiemminkin, mutta takaisinkäännetyn koodin ymmärtäminen on huomattavasti hankalampaa. Toinen merkittävä obfuskoinnista saatava hyöty on tavukooditiedostojen koon pieneneminen, sillä obfuskoinnissa kutistetaan koodin pakkausten, luokkien, metodien ja muuttujien nimiä. Tämän vuoksi obfuskointi on erittäin käytännöllinen tekniikka esimerkiksi käytettäviltä muistimääriltään rajoitetuissa Java-mobiilisovelluksissa. Obfuskoinnin haittapuolia ovat tavukoodin rakenteen muuttumisesta mahdollisesti aiheutuvat toiminnalliset ongelmat ja tavukoodiin lisättävän ylimääräisen sotkun aiheuttama sovelluksen suoritusnopeuden heikkeneminen. [10; 11.]

Nykyään käytössä olevat obfuskointitekniikat ovat *Name Obfuscation* (josta käytetään myös nimitystä *Layout Obfuscation*), *Flow Obfuscation* (josta käytetään myös nimitystä *Control Obfuscation*), *String Encryption* ja *Line Number Scrambling*. Useat obfuskointityökalut käyttävät vain joitakin näistä tekniikoista, mutta parhaan suojaustason saavuttaa käyttämällä kaikkia tekniikoita.

3.3.1 *Name Obfuscation -tekniikka*

Ensimmäinen kehitetty obfuskointitekniikka, *Name Obfuscation*, muuttaa pakkausten, luokkien, metodien ja muuttujien nimiä tunnistamattomaan muotoon [10]. Esimerkiksi pakkauspolun com.mycompany.packet1.Class1 nimi voi *Name Obfuscationin* jälkeen muuttua muotoon a.b.c.d. Tällaisen muutoksen ansiosta pakkausten, luokkien, metodien ja muuttujien nimet ovat lyhyempiä, ja siten vähemmän levytilaa vieviä kuin alkuperäiset nimet. Muutettuina nimet eivät myöskään anna takaisinkääntäjälle mitään tietoa sisällöistään.

3.3.2 *Flow Obfuscation -tekniikka*

Useat nykyisetkin obfuskointityökalut käyttävät pelkkää *Name Obfuscation* -tekniikkaa suojauskeinonaan. Tämä tekniikka ei kuitenkaan sellaisenaan riitä suojauskeinoksi, sillä se ei sotke ohjelman varsinaista toimintalogiikkaa mitenkään. Tähän tarjoaa ratkaisun uudempi *Flow Obfuscation* -tekniikka.

Flow Obfuscationin tarkoitus on sotkea koodin toimintalogiikka ja rakenne tunnistamattomaan muotoon [11]. Tämä tapahtuu muun muassa lisäämällä turhaa käyttämätöntä koodia käytetyn koodin joukkoon sekä muuttamalla koodin rakenteiden syntaksia ja suoritusjärjestystä. Esimerkki tällaisesta suoritusjärjestyksen muuttamisesta on tilanne, jossa silmukkarakenteen suoritus käännetään tapahtuvaksi takaperin. Parhaimmillaan *Flow Obfuscation* tekee takaisinkäännetystä tavukoodista erittäin vaikeasti ymmärrettävää. [10.]

3.3.3 *String Encryption -tekniikka*

String Encryption -tekniikka nimensä mukaisesti sotkee merkkijonojen sisällön käyttämällä enkryptausta [12]. Kuten koko tavukoodin enkryptauksen tapauksessa, myös pelkkien merkkijonojen enkryptaus saattaa hidastaa sovelluksen toimintaa ja kasvattaa tavukooditiedostojen kokoa. Tällöin puhutaan kuitenkin erittäin pienistä muutoksista, toisin kuin koko tavukoodin enkryptauksen tapauksessa.

3.3.4 *Line Number Scrambling- ja Stack Trace Translation -tekniikat*

Line Number Scrambling -tekniikan ideana on sotkea tavukoodista löytyvät lähdekoodia vastaavat rivinumerot. Yleensä ilman *Line Number Scrambling* -tekniikkaa toimivat obfuskointityökalut poistavat nämä rivinumerot tavukoo-

dista kokonaan antaakseen takaisinkääntäjälle vähemmän tietoa lähdekoodista. Tällöin ongelmaksi kuitenkin muodostuvat Javan Stack trace -ilmoitukset, joissa koodin aiheuttaman virheen jälkeen näkyy mille riville tämä virhe viittaa. Mikäli rivinumerot ovat poistettu tavukoodista obfuskoinnin myötä, Stack trace -ilmoitukset viittaavat vain tuntemattomille rivinumeroille. Tästä syystä *Line Number Scrambling* on erittäin käytännöllinen obfuskointitekniikka käytettynä yhdessä *Stack Trace Translation* -tekniikan kanssa. [13.]

Stack Trace Translationin avulla obfuskoidun sovelluksen tuottaman Stack trace -ilmoituksen voi kääntää vastaamaan alkuperäistä lähdekoodia [14]. Tämä on tarpeellinen toiminto, koska obfuskoidessa pakkausten, luokkien ja metodien nimet muuttuvat. Tällöin obfuskoidun sovelluksen tuottama Stack trace -ilmoitus viittaa uusiin, obfuskoituihin pakkausten, luokkien ja metodien nimiin. Tällaisessa tilanteessa on tärkeää, että tämä viittaus onnistutaan kääntämään jäljitettävään muotoon vastaamaan alkuperäisen lähdekoodin nimiä.

4 OBFUSKOINTITYÖKALUT VERTAILTAVIKSI TESTISOVELLUKSEN KANSSA

Valmiita obfuskointityökaluja on olemassa lukuisia, niin kaupallisia kuin ilmaisiakin ohjelmistoja. Primavistan Java-koodin obfuskointiprosessin lähtökohtana oli Gordionin toiveiden mukaisesti muutaman olemassa olevan obfuskointityökalun valinta tarkempaa vertailua varten. Vertailun tarkoituksena oli valita paras ja toimivin vaihtoehto, jonka hankintaa ja käyttöönottoa ehdotettaisiin yritykselle.

4.1 Työkalujen valinta tarkempaan vertailuun

Internetin hakukoneita, artikkeleita ja eri Java-aiheisia keskustelupalstoja tutkimalla valmiita obfuskointityökaluja löytyi yhteensä 10 kappaletta. Koska useissa näistä työkaluista oli selviä puutteita, ei käytettävissä olleen ajan, yrityksen toiveiden sekä tämän lopputyön laajuuden kannalta ollut järkevää sisällyttää näitä kaikkia 10 työkalua varsinaiseen vertailuun. Näin ollen laadittiin kriteerit, joilla lopulliseen vertailuun valittavien työkalujen määrä saataisiin rajattua 4 - 6:een.

Työkalun laatua ja yrityksen tarpeita ajatellen valintakriteereiksi otettiin seuraavat seikat:

- Työkalun on oltava edelleen aktiivisesti kehitettävänä.
- Jos työkalu on kaupallinen, siitä on oltava saatavilla ilmainen kokeiluversio.
- Jos työkalu on kaupallinen, se on voitava maksaa kertaostoksena, eikä esimerkiksi vuotuisena lisenssimaksuna.
- Työkalun on oltava käytettävissä myös komentoriviltä, jolloin se on helposti integroitavissa Primavistan Ant-buildiin.
- Työkalussa on oltava mahdollisuus jättää halutut osiot (esimerkiksi tietyt pakkaukset tai luokat koodissa) kokonaan obfuskoimatta.
- Työkalun on osattava kääntää obfuskoidun koodin tuottamat Javan Stack trace -ilmoitukset vastaamaan alkuperäisen obfuskoimattoman koodin vastaavia ilmoituksia.

Haluttujen osioiden obfuskoimatta jättäminen on erittäin tärkeä ohjelman ominaisuus. Tämä korostuu erityisesti niissä tapauksissa, joissa obfuskointi aiheuttaa jonkin koodin osion toimimattomuuden. Obfuskointitekniikan luonteen vuoksi tällaisia toimimattomuutta aiheuttavia tilanteita saattaa syntyä esimerkiksi Javan RMI:tä käyttävän koodin yhteydessä. Tällaisissa tilanteissa on oltava mahdollisuus jättää ongelmia aiheuttavat koodin osiot kokonaan obfuskoimatta. Kaikki obfuskointityökalut eivät tällaista mahdollisuutta tarjoa.

Koska obfuskointi muuttaa ja lyhentää pakkausten, luokkien, metodien ja muuttujien nimiä, obfuskoidun koodin tuottama Stack trace -ilmoitus viittaa silloin aina obfuskoituihin nimiin. Ohjelmoijien on luonnollisestikin erittäin hankalaa lähteä jäljittämään ohjelmassa olevaa virhettä, jos Stack trace -ilmoitus viittaa esimerkiksi sellaiseen pakkaus- ja luokkapolkuun kuin com.gordion.a.b.c.d. Tällaisissa tilanteissa onkin tärkeää, että obfuskointityökalu osaa kääntää obfuskoidun koodin tuottaman Stack trace -ilmoituksen vastaamaan alkuperäisen, obfuskoimattoman koodin pakkaus- ja luokkapolkuja. Myöskään tätä ominaisuutta eivät kaikki obfuskointityökalut tarjoa.

Edellä esitettyjen valintakriteerien perusteella 10 obfuskointityökalusta kuusi seuloitui pois ja jäljelle jäi vain neljä työkalua. Koska alkuperäinen tarkoituskin oli valita tarkempaan vertailuun 4 - 6 työkalua, valittiin nämä jäljelle jää-

neet neljä työkalua vertailtaviksi (taulukko 1). Näistä työkaluista kaksi olivat kaupallisia ohjelmistoja (Zelix KlassMaster ja DashO Pro) ja kaksi ilmaisia ohjelmistoja (yGuard ja ProGuard).

Taulukko 1. Vertailtaviksi valitut obfuskointityökalut ominaisuuksineen [15, 16, 17, 18]

Obfuskoija	**Käyttöoikeus**	**Ilmoitetut obfuskointitekniikat**
Zelix Klassmaster v5.0	Kokeiluversio (30 päivää)	Name Obfuscation, Flow Obfuscation, String Encryption, Line Number sSrambling
ProGuard v4.0 (beta)	Ilmainen (GPL-lisenssin alainen)	Name Obfuscation
yGuard v2.2.0	Ilmainen (freeware)	Name Obfuscation, Line Number Scrambling
DashO Pro v3.3	Kokeiluversio (14 päivää)	Name Obfuscation, Flow Obfuscation, String Encryption

Lisäksi valittiin yksi takaisinkääntäjäohjelma (*decompiler*), jota oli tarkoitus käyttää takaisinkääntämään obfuskointityökalujen obfuskoimaa tavukoodia. Kuten obfuskointityökaluja, myös valmiita takaisinkääntäjiä löytyy runsaasti internetistä. Tätä työtä varten takaisinkääntäjäksi valittiin suosittu DJ Java Decompiler v3.7, joka on ilmainen ja helppokäyttöinen freeware-ohjelma.

4.2 Testisovellus

Vertailuun valittuja obfuskointityökaluja vertailtiin Javalla itse ohjelmoidun testisovelluksen sekä valitun takaisinkääntäjän avulla. Tarkoituksena oli, että testisovelluksen tavukooditiedostot ensin takaisinkäännettiin valittua takaisinkääntäjää käyttäen. Tällä varmistettiin, että takaisinkääntäjäohjelma todella toimi ja että takaisinkäännettyä lähdekoodia voitiin myös käyttää uudelleen. Tämän jälkeen testisovelluksen tavukooditiedostot obfuskoitiin ja niitä yritettiin uudelleen takaisinkääntää. Näin toimittaessa nähtiin, kuinka hyvin kukin obfuskointityökalu osaa obfuskoida koodia ja miten niiden tekemä obfuskointi haittaa takaisinkääntämistä.

Testisovellus sisälsi kaksi luokkaa (Class1 ja Class2), sekä niistä tehdyn JAR-paketin. Koodissa käytettiin seuraavia sopivia rakenteita ja ominaisuuksia eri obfuskointitekniikoiden testaamiseksi:

- silmukoita ja if-else-rakenteita *Flow Obfuscation* -tekniikan testaamiseksi

- metodikutsuja ja luokkien välillä toimivia olioita *Name Obfuscation* -tekniikan testaamiseksi
- merkkijonojen käyttöä *String Encryption* -teknikan testaamiseksi
- erilaisia muuttujien ja metodien suojausmääreitä niiden säilymisen testaamiseksi
- manuaalisen poikkeuksen (*exception*) tekeminen (*Stack Trace Translation* -tekniikan testaamiseksi).

Luokassa Class2 on muuttujana arvottu kokonaisluku (int) 1 - 10. Luokan konstruktorissa tämä kyseinen arvottu luku sijoitetaan toiseen kokonaislukumuuttujaan for-silmukassa. Tämän jälkeen kyseinen luku ja tieto luvun suuruudesta lähetetään merkkijonomuotoisena parametrina Class1:n setString-metodille. Tieto luvun suuruudesta määräytyy if-rakenteen mukaisesti. Tämän jälkeen kutsutaan Class1:n printString-metodia, joka tulostaa konsolille merkkijonon, jossa on kyseinen arvottu luku sekä tieto siitä, onko kyseinen luku pienempi, suurempi vai yhtä suuri kuin 5. Lopuksi tehdään manuaalinen poikkeus, NullPointerException.

Luokassa Class1 on sovelluksen main-metodi, jossa luodaan olio luokasta Class2, sekä toteutukset metodeille setString, getString ja printString. Metodi setString saa parametrinaan merkkijonon, joka tässä metodissa sijoitetaan luokan sisäiseen muuttujaan. Metodi getString palauttaa tämän muuttujan. Metodi printString puolestaan tulostaa konsolille getString-metodilta saamansa muuttujan sisällön eli luokan Class2 tuottaman merkkijonon.

Testisovelluksen ohjelmakoodi on esitetty seuraavassa:

```
/*
 * Class "Class1":
 * - Tests Name Obfuscation and String Encryption
 * - Tests the correct working of the application
 *
 * @author Vesa Herranen
 * @date 25.6.2007
 */

public class Class1 {

   private String finalString;

   public void setString(String str) {
      finalString = str;
   }

   protected String getString() {
      return finalString;
```

```
   }

   public void printString() {
      System.out.println(getString()+"\n");
   }

   public static void main(String[] args) {
      Class2 class2 = new Class2();
   }
}


/*
 * Class "Class2":
 *  - Tests Flow Obfuscation and Stack Trace Translation
 *
 * @author Vesa Herranen
 * @date 25.6.2007
 */

import java.util.Random;

public class Class2 {
   private Class1 class1 = new Class1();
   private Random randomGenerator = new Random();
   private int randomInt = randomGenerator.nextInt(10);
   private int number = 0;
   private String numberStr = "Random number: ";

   public Class2() {
      for (int i=0; i < randomInt; i++)
         number++;

      if (number < 5)
         class1.setString(numberStr + number + " < 5");
      else if (number == 5)
         class1.setString(numberStr + number + " = 5");
      else
         class1.setString(numberStr + number + " > 5");

      class1.printString();
      throw new NullPointerException(
         "Manually created exception");
   }
}
```

Ajettuna testisovellus tulostaa:

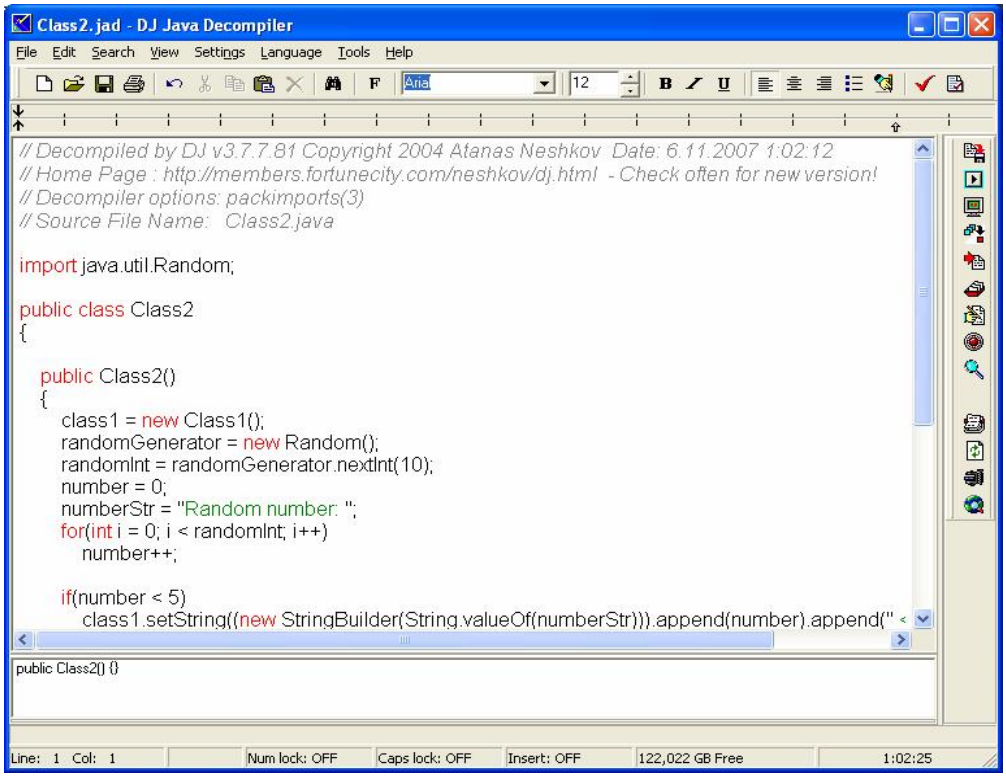
```
Random number: 2 < 5
Exception in thread "main" java.lang.NullPointerException:
Manually created exception
      at Class2.<init>(Class2.java:30)
      at Class1.main(Class1.java:27)
```

4.3 Testisovelluksen koodin takaisinkääntäminen

Testisovelluksen koodin takaisinkääntämiseen käytettiin edellä mainittua DJ Java Decompileria, joka on helppokäyttöinen graafinen työkalu Javan tavukooditiedostojen takaisinkääntämiseen. Ohjelmalla voidaan takaisinkääntää

joko pelkkiä yksittäisiä tavukooditiedostoja tai kaikki esimerkiksi JAR-pakkauksen sisältämät tavukooditiedostot.

Testisovelluksen kummankin tavukooditiedoston takaisinkääntäminen sujui täysin ongelmitta (kuva 3). Takaisinkäännetty koodi (liite 1) oli selkeää, ymmärrettävää ja lähes täysin samannäköistä kuin alkuperäinenkin koodi. Ajettuna takaisinkäännetty koodi toimi kuten kuuluukin. Johtopäätöksenä voitiin siis todeta, että valittu takaisinkääntäjäohjelma osaa takaisinkääntää Javan tavukooditiedostoja ongelmitta. Näin ollen sitä voitiin käyttää apuna obfuskointityökaluja testatessa.



Kuva 3. DJ Java Decompilerin pääikkuna, jossa takaisinkäännettynä testisovelluksen luokka Class2

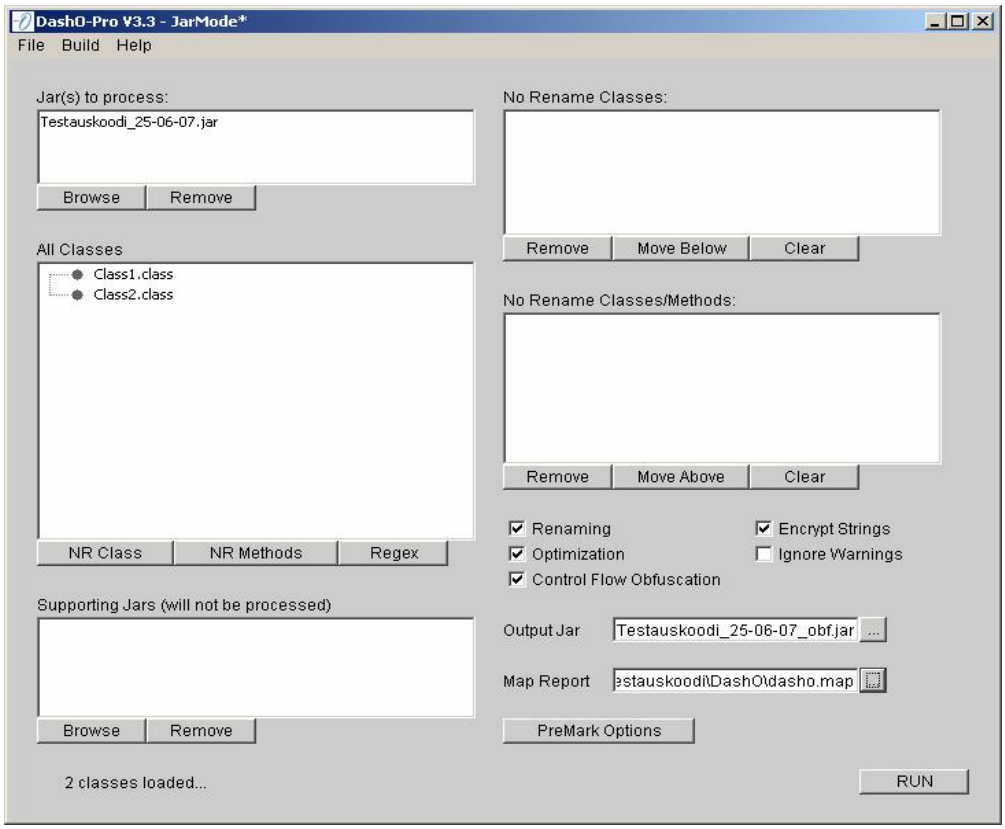
5 OBFUSKOINTITYÖKALUJEN VERTAILU

Kaikkia vertailuun valittua neljää obfuskointityökalua testattiin testisovelluksen koodin sekä valitun takaisinkääntäjän avulla. Tämä tapahtui edellä kuvaillun mukaisesti siten, että testisovelluksen tavukooditiedostot obfuskoitiin, jonka jälkeen niitä yritettiin takaisinkääntää. Näin toimittaessa nähtiin, miten kukin obfuskointityökalu osaa obfuskoida koodia ja miten niiden tekemä obfuskointi haittaa takaisinkääntämistä. Erityishuomiota kiinnitettiin myös Stack

trace -ilmoitusten kääntämiseen, sillä tämä jokaisen vertailuun valitun työkalun tarjoama ominaisuus saattaa joskus olla vaikeakäyttöinen.

5.1 DashO-ohjelma

Testaussovelluksen koodin obfuskointi DashO:lla onnistui ongelmitta, eikä obfuskoitua JAR-pakettia ajettaessa sovelluksen toiminta ollut muuttunut mitenkään. Huomattavaa kuitenkin on, että DashO käyttää automaattisesti niin sanottua aggressiivista obfuskointia. Tämä tekniikka toistuvasti uudelleennimeää muuttujia, metodeja, luokkia ja pakkauksia samoilla nimillä vaikeuttaakseen entistä enemmän obfuskoidun koodin ymmärtämistä. Vaikka kyseinen tekniikka useimmissa tapauksissa tuottaisikaan ongelmia, sen käyttöä ei aina pidetä kovin suotavana. Tämä johtuu siitä, että joskus sovelluksia ajettaessa samannimisyydet saattavat tuottaa ongelmia. Tätä ominaisuutta ei myöskään saanut DashO:sta pois päältä.



Kuva 4. DashO:n pääikkuna valitulla testisovelluksen JAR-paketilla

DashO:n obfuskoimien tavukooditiedostojen takaisinkääntäminen onnistui. Takaisinkäännettyä koodia (liite 2) tarkasteltaessa huomattiin, että muuttujien, metodien ja toisen luokan nimet olivat erilaiset ja koodissa oli mukana

suuri määrä ylimääräistä käsittämätöntä tekstiä. Myös kaikkien merkkijonojen sisällöt olivat muuttuneet käsittämättömiksi *String Encryption* -tekniikan ansiosta. Koko ohjelman toimintalogiikka oli muuttunut käsittämätöksi ilman syvempää tutkimista *Flow Obfuscation* -tekniikan myötä, joskin alkuperäisen koodin tuntemalla pieni logiikka oli havaittavissa.

Takaisinkäännetty koodi ei kuitenkaan enää mennyt edes kääntäjästä läpi, koska koodi sisälsi neljä kappaletta virheitä. Näin ollen takaisinkäännettyä koodia ei pystynyt ajamaankaan.

Seuraavassa on esitetty testisovelluksen oleellisimman toimintalogiikan sisältävän luokan Class2 konstruktorin DashO:n obfuskoima tavukoodi takaisinkäännettynä:

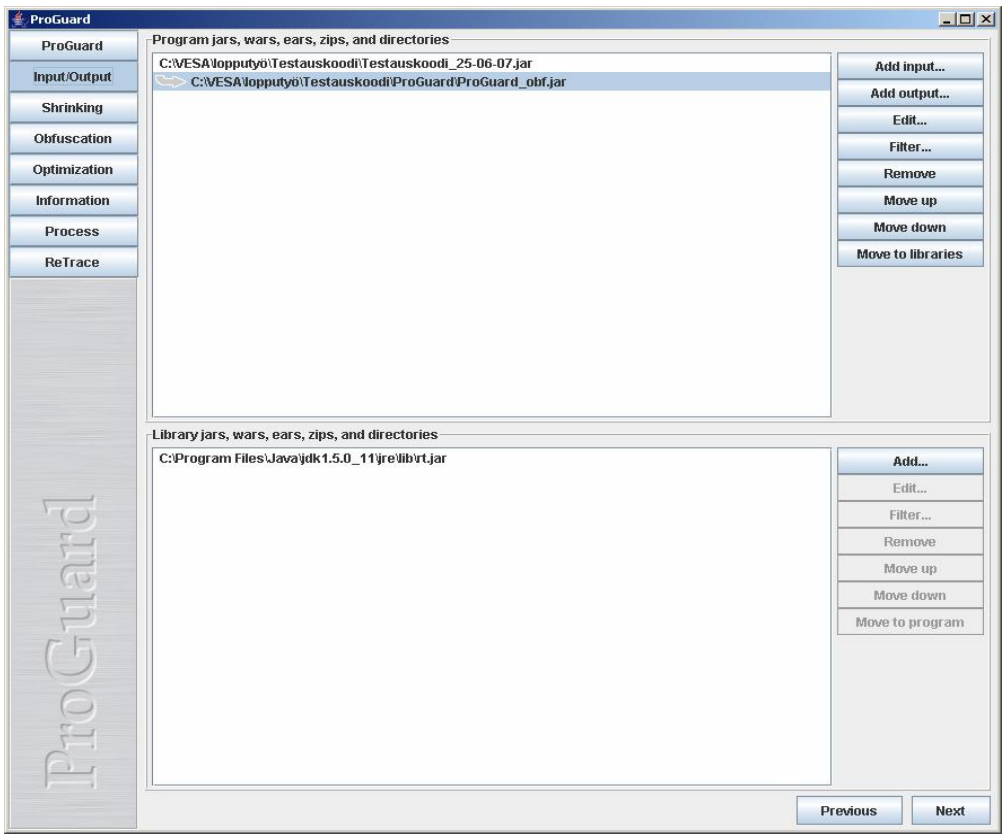
```
public b()
{
    a = new Class1();
    DashoEval_b = new Random();
    DashoEval_c = DashoEval_b.nextInt(10);
    DashoEval_d = 0;
    DashoEval_e = Class1.A_B_C_D("Samhli'j~eio\1778/");
    break MISSING_BLOCK_LABEL_54;
    _L4:
       a.DashoEval_b();
       throw new
          NullPointerException(Class1.A_B_C_D(
             "Namwfjk\177)kyonzll1ukqrfc\177xx"));
    _L2:
       while(DashoEval_d != 5)
       {
          a.a((new StringBuilder(
             String.valueOf(DashoEval_e))).append(
                DashoEval_d).append(Class1.A_B_C_D(
                   "!@#7")).toString());
          continue; /* Loop/switch isn't completed */
       }
       a.a((new StringBuilder(String.valueOf(
          DashoEval_e))).append(DashoEval_d).append(
             Class1.A_B_C_D("!=#7")).toString());
       continue; /* Loop/switch isn't completed */
       for(int i = 0; i < DashoEval_c; i++)
          DashoEval_d++;

       if(DashoEval_d >= 5) goto _L2; else goto _L1
    _L1:
       a.a((new StringBuilder(String.valueOf(
          DashoEval_e))).append(DashoEval_d).append(
             Class1.A_B_C_D("!>#7")).toString());
       if(true) goto _L4; else goto _L3
    _L3:
}
```

Stack trace -ilmoituksen kääntäminen viittaamaan alkuperäiseen koodiin pitäisi DashO:lla onnistua sen tuottaman map-tiedoston ja ohjelman graafisen Translator-työkalun avulla. Testisovelluksen tekemää Stack trace -ilmoitusta ei kuitenkaan onnistuttu kääntämään, sillä DashO ilmoitti virheilmoituksen "Failed to translate stack trace: java.lang.NullPointerException". Kaikki tarvittavat toiminnot oli kuitenkin tehty oikein, joten jäi epäselväksi, miksei DashO:n Stack trace -käännös onnistunut.

5.2 ProGuard-ohjelma

Kuten DashO:lla, myös Proguardilla tehty testaussovelluksen koodin obfuskointi onnistui ongelmitta. Tässäkään tapauksessa testisovelluksen obfuskoitua JAR-pakettia ajettaessa sovelluksen toiminta ei ollut muuttunut miten-kään.



Kuva 5. ProGuardin pääikkuna valitulla testisovelluksen JAR-paketilla

Myös ProGuardin obfuskoimien tavukooditiedostojen takaisinkääntäminen onnistui. Takaisinkäännetyssä koodissa (liite 3) muuttujien, metodien ja toisen luokan nimet olivat muuttuneet. Ohjelman toimintalogiikka oli kuitenkin täysin muuttumaton, sillä ProGuard ei käytä ollenkaan *Flow Obfuscation*

-tekniikkaa. Takaisinkäännetty koodi meni kääntäjästä läpi ja ajettaessa sen toiminta oli täysin oikeanlaista.

Seuraavassa on esitetty luokan Class2 konstruktorin ProGuardin obfuskoima tavukoodi takaisinkäännettynä:

```
public a()
{
   a = new Class1();
   b = new Random();
   c = b.nextInt(10);
   d = 0;
   e = "Random number: ";
   for(int i = 0; i < c; i++)
      d++;

      if(d < 5)
         a.a((new StringBuilder(



               String.valueOf(e))).append(d).append(
                  " < 5").toString());
      else
      if(d == 5)
         a.a((new StringBuilder(
            String.valueOf(e))).append(d).append(
               " = 5").toString());
      else
         a.a((new StringBuilder(
            String.valueOf(e))).append(d).append(
               " > 5").toString());
      a.a();
      throw new NullPointerException(
         "Manually created exception");
}
```

Stack trace -ilmoituksen kääntäminen viittaamaan alkuperäiseen koodiin onnistui ongelmitta ProGuardin tuottaman map-tiedoston avulla. Kääntämisessä käytettiin ohjelman graafista ReTrace-ominaisuutta. Koodissa käytetty Javan RMI-tekniikka saattaa lakata obfuskoinnin myötä toimimasta. Tästä syystä ProGuard tarjoaa toiminnon, joka automaattisesti ottaa RMI-koodia sisältävät luokat huomioon. Tällöin ne obfuskoidaan siten, että koodi varmasti toimii obfuskoinnin jälkeenkin. [16.]

5.3 yGuard-ohjelma

yGuard toimii ainoastaan Ant-buildiin integroituna, eikä se tarjoa graafista käyttöliittymää muuhun kuin Stack trace -ilmoitusten kääntämiseen. Jotta yGuardia voitiin käyttää testisovelluksen koodin obfuskoimiseen, piti testisovellukselle tehdä XML-muotoinen Ant-build (tiedosto build.xml). Siinä kutsut-

tiin yGuardin JAR-pakettia halutuilla parametreilla, jotka vastaavat prosessissa käytettäviä asetuksia. Obfuskointi onnistui ongelmitta ajamalla tämä build-tiedosto Eclipsessä.

Ajon jälkeen konsolille tulostui seuraavat tiedot obfuskoinnista:

```
Buildfile: C:\project\work\eclipse\ObfuskointiTesti\build.xml
init:
compile:
jar:
      [jar] Building jar:
      C:\project\work\eclipse\ObfuskointiTesti\
      ObfuskointiTesti.jar
yguard:
      [shrink] yGuard Shrinker v2.2.0 -
      http://www.yworks.com/products/yguard
      [shrink] no entrypoints given - using class access
      public and protected on all inoutpairs.
      [shrink] parsing
      C:\project\work\eclipse\ObfuskointiTesti\
      ObfuskointiTesti.jar
      [shrink] writing shrinked
      C:\project\work\eclipse\ObfuskointiTesti\
      ObfuskointiTesti.jar to
      C:\project\work\eclipse\ObfuskointiTesti\
      yguard_temp_47324.jar.
      [shrink] shrinked
      C:\project\work\eclipse\ObfuskointiTesti\
      ObfuskointiTesti.jar BY 3,32%.
      [shrink] size before: 1 KB, size after: 1 KB.
      [shrink] removed 0 classes, 0 methods, 0 fields, 0
      resources.
      [shrink] 2 classes remaining of 2 total.
      [rename] yGuard Obfuscator v2.2.0 -
      http://www.yworks.com/products/yguard
      Parsing jar C:\project\work\eclipse\ObfuskointiTesti\
      yguard_temp_47324.jar
      Obfuscating Jar
      C:\project\work\eclipse\ObfuskointiTesti\
      yguard_temp_47324.jar to ObfuskointiTesti_obf.jar
      BUILD SUCCESSFUL
      Total time: 3 seconds
```

Obfuskoitua testisovelluksen JAR-pakettia ajettaessa sovelluksen toiminta ei ollut tässäkään tapauksessa muuttunut. Takaisinkäännetyn koodin (liite 4) muuttujien, metodien ja toisen luokan nimet olivat muuttuneet. Kuten ProGuardin tapauksessa, myös tässä tapauksessa ohjelman toimintalogiikka oli täysin muuttumaton, sillä myöskään yGuard ei käytä ollenkaan Flow Obfuscation -tekniikkaa. Takaisinkäännetty koodi (liite 4) meni kääntäjästä läpi ja ajettaessa sen toiminta oli täysin oikeanlaista.

Seuraavassa on esitetty luokan Class2 konstruktorin yGuardin obfuskoima tavukoodi takaisinkäännettynä.

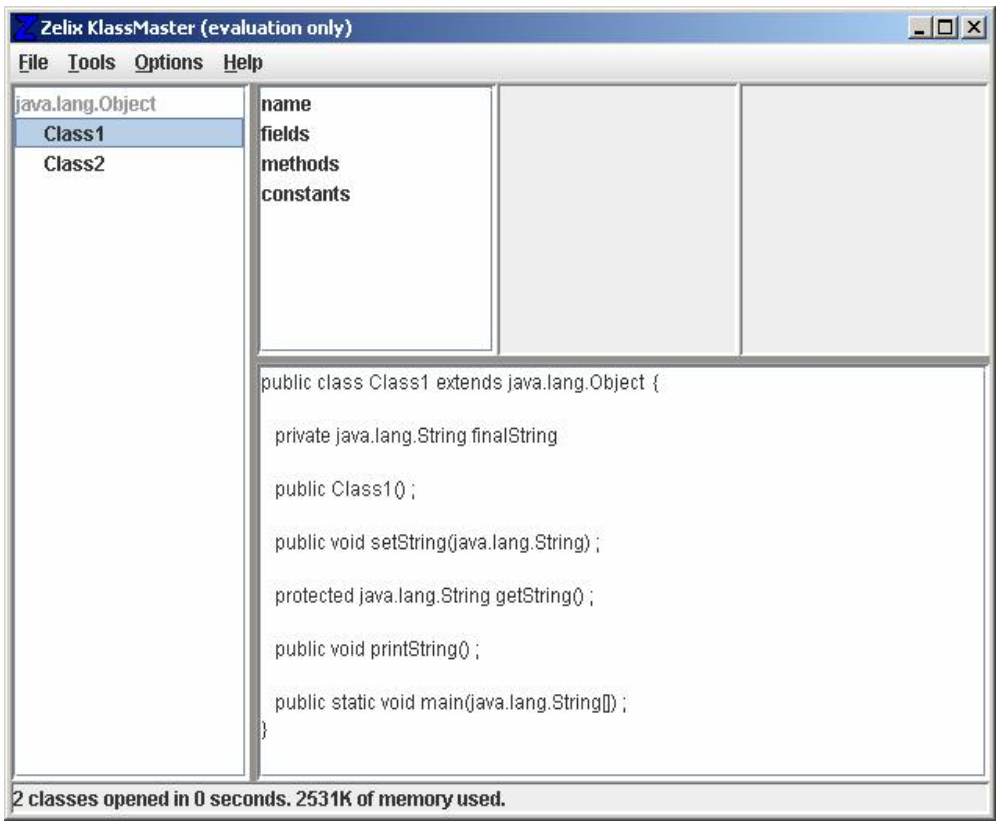
```
public A()
 {
    A = new Class1();
    D = new Random();
    C = D.nextInt(10);
    B = 0;
    E = "Random number: ";
    for(int i = 0; i < C; i++)
       B++;

    if(B < 5)
       A.A((new StringBuilder(
          String.valueOf(E))).append(B).append(
             " < 5").toString());
    else
    if(B == 5)
       A.A((new StringBuilder(
          String.valueOf(E))).append(B).append(
             " = 5").toString());
    else
       A.A((new StringBuilder(
          String.valueOf(E))).append(B).append(
             " > 5").toString());
    A.B();
    throw new NullPointerException(
       "Manually created exception");
}
```

Stack trace -ilmoituksen kääntäminen viittaamaan alkuperäiseen koodiin onnistui ongelmitta yGuardin tuottaman map-tiedoston avulla. Kääntämisessä käytettiin ohjelman graafista Element Mapping-työkalua. DashO:sta ja ProGuardista poiketen yGuard tarjoaa lisäksi *Line Number Scrambling* -valinnan rivinumeroiden sotkemiseksi.

5.4 Zelix KlassMaster -ohjelma

Kuten kaikilla muillakin obfuskointityökaluilla, myös Zelix KlassMasterilla tehty obfuskointi onnistui ongelmitta. Obfuskoitua JAR-pakettia ajettaessa tuli kuitenkin virheilmoitus "Exception in thread "main" java.lang.NoSuchMethodError: main". Tämä ilmoitus viittaa siihen, ettei koodista löydy main-metodia, joka käynnistäisi sovelluksen. Ongelma saatiin korjattua valitsemalla KlassMasterin asetuksista, ettei main-metodin nimeä lainkaan obfuskoida, vaan se jätetään muuttumattomaksi. Tämän jälkeen obfuskoidun JAR-paketin ajaminen onnistui, eikä sovelluksen toiminta ollut muuttunut mitenkään.



Kuva 6. Zelix KlassMasterin pääikkuna valitulla testisovelluksen JAR-paketilla

Kaikkien muiden obfuskointityökalujen tapaan myös Zelix KlassMasterin obfuskoimien tavukooditiedostojen takaisinkääntäminen onnistui. Takaisinkäännetyssä koodissa (liite 5) muuttujien, metodien ja kummankin luokan nimet olivat muuttuneet. Kaikkien merkkijonojen sisällöt olivat muuttuneet käsittämättömiksi String Encryption -tekniikan vuoksi, kuten DashO:n tapauksessakin. Samoin koko ohjelman toimintalogiikka oli muuttunut todella epäselväksi Flow Obfuscation -tekniikan myötä.

Takaisinkäännetty koodi ei myöskään mennyt kääntäjästä läpi, sillä koodi sisälsi 22 virhettä. Näin ollen takaisinkäännettyä koodia ei pystynyt myöskään ajamaan. Seuraavassa on esitetty luokan Class2 konstruktorin Zelix KlassMasterin obfuskoima tavukoodi takaisinkäännettynä:

```
public b()  {
    int i;
    int j;
    j = a.b;
    super();
    a = new a();
    b = new Random();
    c = b.nextInt(10);
    d = 0;
    e = z[4];
    i = 0;
```

```
    if(j == 0) goto _L2; else goto _L1
    _L1:
       d++;
       if(true)
       continue; /* Loop/switch isn't completed */
       d;
       1;
       JVM INSTR iadd ;
       d;
       i++;
    _L2:
       if(i < c) goto _L1; else goto _L3
    _L3:
       this;
       if(j != 0) goto _L5; else goto _L4
    _L5:
       this;
    _L4:
       d;
       5;
       JVM INSTR icmpge 139;
       goto _L6 _L7
    _L6:
       break MISSING_BLOCK_LABEL_96;
    _L7:
       break MISSING_BLOCK_LABEL_139;
       a.a((new StringBuilder(
          String.valueOf(e))).append(d).append(
             z[3]).toString());
       if(j == 0)
          break MISSING_BLOCK_LABEL_229;
       if(d == 5)
       {
          a.a((new StringBuilder(
             String.valueOf(e))).append(d).append(
                z[0]).toString());
          if(j == 0)
             break MISSING_BLOCK_LABEL_229;
       }
       a.a((new StringBuilder(
          String.valueOf(e))).append(d).append(
             z[2]).toString());
       a.b();
       throw new NullPointerException(z[1]);
}
```

Stack trace -ilmoituksen kääntäminen viittaamaan alkuperäiseen koodiin onnistui ongelmitta KlassMasterin tuottaman changelog-tiedoston avulla. Kääntämisessä käytettiin ohjelman graafista Stack Trace Translation -työkalua. yGuardin tavoin myös Zelix KlassMaster tarjoaa *Line Number Scrambling* -valinnan rivinumeroiden sotkemiseksi. Kuten ProGuard, myös Zelix KlassMaster osaa automaattisesti huomioida RMI-koodia sisältävät luokat obfuskointia tehdessään. [15.]

5.5 Työkalujen vertailun tulokset

Eniten obfuskointitekniikoita – yhteensä viisi eri tekniikkaa – tarjoaa Zelix KlassMaster. Toinen kaupallinen työkalu, DashO, tarjoaa neljä eri tekniikkaa. Ilmaiset työkalut ProGuard ja yGuard tarjoavat ainoastaan kaksi tekniikkaa (taulukko 1).

Valittua takaisinkääntäjää vastaan selvästi parhaiten menestyi Zelix KlassMaster, jonka obfuskoima tavukoodi takaisinkäännettynä oli käsittämätöntä ilman koodin perusteellisempaa tarkastelua. Koodi ei myöskään mennyt kääntäjästä läpi suuren virhemäärän vuoksi. Seuraavaksi parhaiten takaisinkääntäjää vastaan menestyi DashO, jonka obfuskoima tavukoodi takaisinkäännettynä oli myös erittäin vaikeaselkoista. Koodista kuitenkin löytyi pieni logiikka sen tuntevalle. Myöskään DashO:n tapauksessa koodi ei mennyt enää kääntäjästä läpi neljän virheen vuoksi. Huonoimmin takaisinkääntäjää vastaan menestyivät ProGuard ja yGuard, jotka ainoastaan muuttivat muuttujien, metodien ja luokkien nimet, mutta jättivät itse koodin toimintalogiikan täysin muuttumattomaksi. Näiden kahden työkalun tapauksessa koodi meni sellaisenaan kääntäjästä läpi.

Varsinainen vertailtujen obfuskointityökalujen pisteytys ja sitä kautta arvottaminen paremmuusjärjestykseen tehtiin työkalujen toteuttamien ominaisuuksien ja toimintojen mukaisesti. Kukin ominaisuus tai toiminto oli yhden pisteen arvoinen. Pisteitä annettiin seuraavista kohdista:

- obfuskoidun testisovelluksen toiminta normaalia
- takaisinkääntäjä ei osaa kääntää obfuskoitua koodia toimivaksi koodiksi
- käyttää toimivasti Name Obfuscation -tekniikkaa
- käyttää toimivasti Flow Obfuscation -tekniikkaa
- käyttää toimivasti String Encryption -tekniikkaa
- käyttää toimivasti Line Number Scrambling -tekniikkaa
- Stack trace -käännökset toimivat.

ProGuard toteutti näistä kohdista ensimmäisen, kolmannen ja viimeisen, saaden yhteispistemääräkseen kolme. yGuard toteutti kohdista ensimmäisen, kolmannen ja kaksi viimeistä, jolloin sen yhteispistemäärä oli neljä. DashO toteutti kaikki muut, paitsi kaksi viimeistä kohtaa, jolloin sen yhteispistemäärä oli viisi. Suurimman pistemäärän kerännyt Zelix KlassMaster toteutti kaikki seitsemän kohtaa, joten se oli vertailun selvä voittaja.

6 ZELIX KLASSMASTERIN TOIMINTA PRIMAVISTAN KANSSA

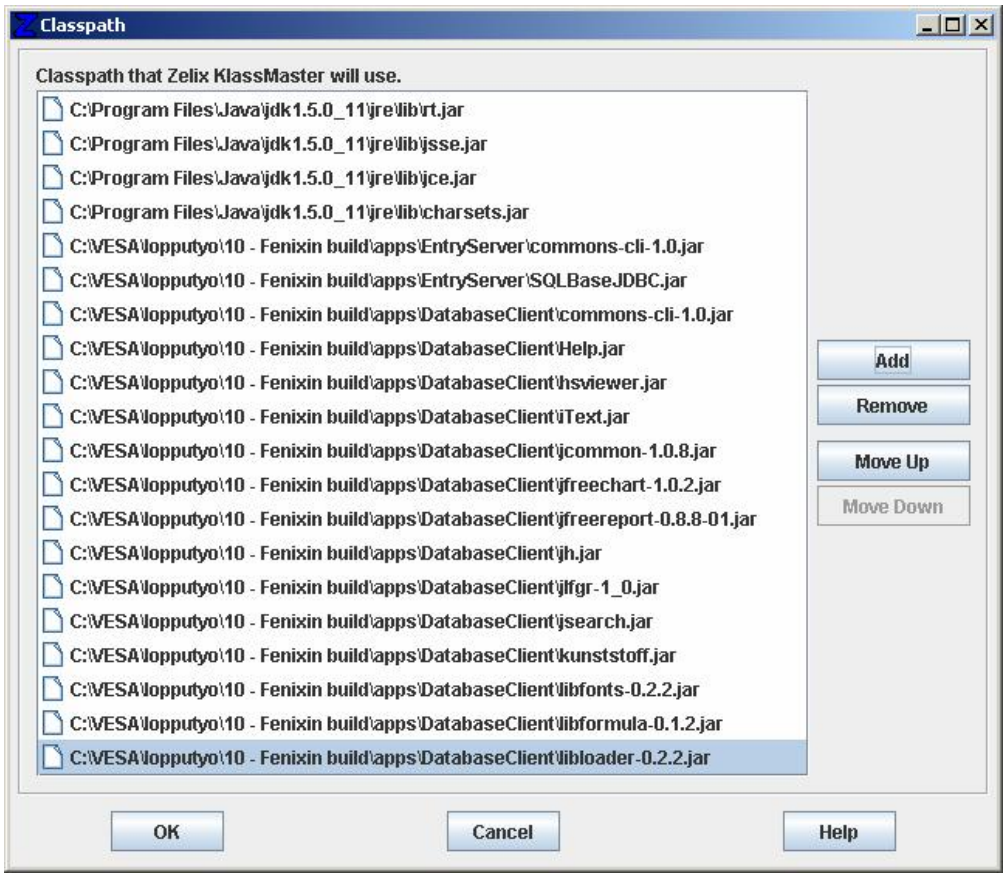
Edellä esitetyn vertailun pohjalta Gordionille päätettiin hankkia obfuskointityökaluksi kaupallinen Zelix KlassMaster. Ohjelmasta päätettiin ostaa lisenssi jokaiselle yrityksen tuotekehitystiimin ohjelmoijalle käytettäväksi. Tällöin obfuskoinnin ajaminen ja tarvittaessa Stack trace -käännökset voitaisiin tehdä jokaisen ohjelmoijan koneella paikallisena toimenpiteenä. Vastaavasti, jos oltaisiin päädytty hankkimaan vain yksi lisenssi kaikkien yhteiskäyttöön, obfuskoinnin ajaminen ja Stack trace -käännökset pitäisi tehdä aina palvelinkoneen kautta.

Ennen ohjelman hankintaa oli kuitenkin tarpeellista tehdä nopea pintapuolinen testaus, jossa Zelix KlassMasterin toiminta testattiin Primavistan koodille. Tuotetta ei siis vielä tässä vaiheessa ostettu.

6.1 Primavistan JAR-pakettien valinta KlassMasterilla

Zelix KlassMasterin testaaminen Primavistalle aloitettiin tekemällä uudet JAR-paketit Primavistan koodista. Tämä tapahtui ajamalla Primavistan valmis Ant-build. Siinä luodaan tavukooditiedostot lähdekoodeista, minkä jälkeen pakataan kaikki tavukooditiedostot kahdeksi JAR-tiedostoksi. Nämä tiedostot ovat asiakaspuolen DatabaseClient.jar ja palvelinpuolen EntryServer.jar. Kyseisten JAR-pakettien takaisinkääntämistä testattiin DJ Java Decompilerilla, jolloin huomattiin, että ne olivat yhtä helposti takaisinkäännettävissä kuin testisovelluksenkin tavukooditiedostot. Tässäkään tapauksessa takaisinkäännettyjen luokkien koodi ei ollut juurikaan muuttunut alkuperäiseen verrattuna.

Kun Zelix KlassMasteria lähdettiin kokeilemaan Primavistan JAR-pakettien kanssa, pyysi ohjelma ensimmäiseksi laittamaan Javan JDK-pakettiin kuuluvien kirjastojen lisäksi kaikki obfuskoitavan sovelluksen kanssa käytetyt apukirjastot Classpath-valintaikkunaansa (kuva 7). Koska Primavistan kanssa käytetään suurta määrää ulkoisia apukirjastoja (JAR-paketteja), oli nämä kaikki muistettava lisätä tässä vaiheessa tähän KlassMasterin Classpath-valintaikkunaan.



Kuva 7. Zelix KlassMasterin Classpath-valintaikkuna Primavistan käyttämillä apukirjastoilla

Seuraavaksi KlassMasterillä valittiin avattaviksi obfuskoimista varten aiemmin tehdyt Primavistan JAR-paketit (DatabaseClient.jar ja EntryServer.jar). Tässä vaiheessa KlassMaster ilmoitti varoituksen koskien kaikkia niitä luokkia, joista löytyy Class.getName()-komentoja, joita ohjelma ei pysty automaattisesti käsittelemään. Tämä johtuu siitä, että obfuskoidessa luokkien nimet muuttuvat niiden alkuperäisistä nimistään. Jos koodiin on kirjoitettu edellä mainitun tyylisiä luokkien nimiin liittyviä komentoja, saattaa obfuskointi aiheuttaa ongelmia. KlassMasterin dokumentaation ohje tällaisiin ongelmiin on jättää Class.getName()-komentojen viittaamien luokkien ja pakkausten nimet obfuskoimatta. Tämä onnistuu KlassMasterin Obfuscate Name Exclusions -valintaikkunassa. [19.]

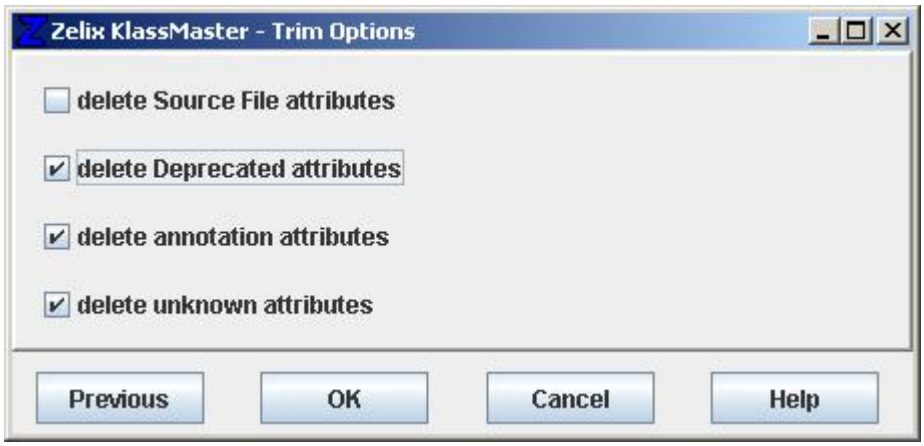
Näiden komentojen toimintaan tulikin kiinnittää erityishuomiota siinä vaiheessa, kun obfuskoidun Primavistan toimintaa testattiin. Tässä vaiheessa näihin komentoihin ei vielä kiinnitetty huomiota.

6.2 Trim-toimenpide

Obfuskointi voitiin aloittaa eri asetuksia käyttäen, kun obfuskoitaviksi halutut Primavistan JAR-paketit oli valittu KlassMasterilla. Ensimmäinen vaihe KlassMasterin tekemässä obfuskoinnissa on sen tarjoama valinnainen Trim-toimenpide. Se on lähes aina obfuskoinnin yhteydessä tehtävä tavukooditiedostojen kutistaminen. Tällä toimenpiteellä pienennetään tavukooditiedostojen kokoa muun muassa poistamalla käyttämättömiä luokkia, metodeja ja muuttujia sekä kääntäjien käyttämiä turhia tietorakenteita [19]. Trim ajettiin läpi jättämättä mitään pakkauksia, luokkia tai metodeita kutistamatta.

6.2.1 Trim-asetukset

Trimin suorituksessa käytettiin seuraavia asetuksia (kuva 8):



Kuva 8. KlassMasterin Trim Options -ikkuna valituilla asetuksilla

Valitsematta jätetty *delete Source File attributes* -valinta viittaa lähdekooditiedostojen tietoihin. Tällaisia tietoja ovat esimerkiksi tavukooditiedostojen lähdekoodiin viittaavat rivinumerot, jotka saattaisivat poistua tämän valinnan myötä. Tällöin Stack trace -ilmoitusta ei enää pystyttäisi jäljittämään oikealle lähdekoodin riville. [19.]

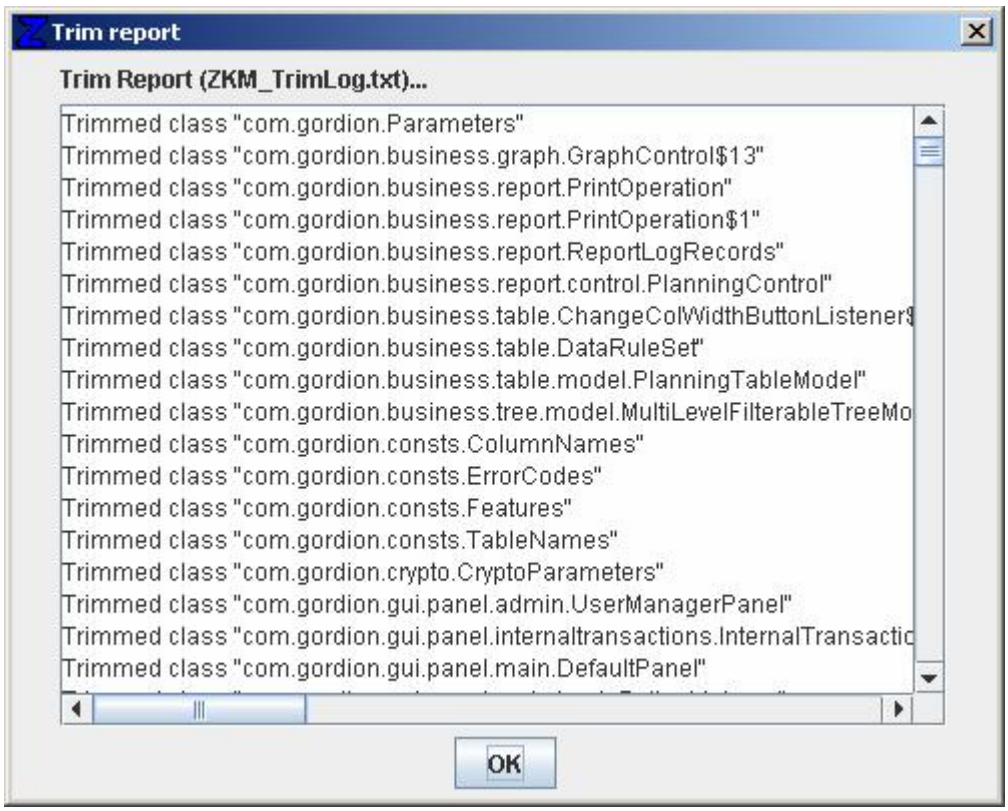
Valittu *delete Deprecated attributes* -valinta poistaa ainoastaan kääntäjien käyttämiä tietoja *deprecated*-tiedolla varustetuista metodeista [19]. *Deprecated* tarkoittaa, että jokin metodi on poistettu käytöstä, jolloin kääntäjät osaavat useimmiten näyttää varoituksen sellaista metodia käytettäessä. *Deprecated*-tieto voi olla valmiiksi määritelty jonkin Javan lähdekoodin metodin yhteyteen, tai sen voi itse lisätä jonkin itse kirjoitetun metodin yhteyteen.

delete annotation attributes -valinta poistaa kaikki – yleensä pelkästään kääntäjän tulkitsemat ja käyttämät – annotaatiot koodista [19]. Annotaatioita eli @-merkin jälkeen kirjoitettavia lauseita käytetään Javassa useimmiten ja Primavistan tapauksessa pelkästään kertomaan jotain asioita kääntäjälle, kuten esimerkiksi metodin olevan *deprecated* [20]. Toisaalta annotaatioita voidaan joskus käyttää myös ajonaikaiseen toimintaan vaikuttavassa tarkoituksessa, kuten Javan EJB (Enterprise Java Beans) 3.0 -tekniikan tapauksessa [21]. Näissä tapauksissa KlassMaster osaa kuitenkin automaattisesti jättää annotaatiot poistamatta, vaikka tämä valinta olisikin valittu [19]. Tästä syystä annotaatioitakaan ei obfuskoinnin jälkeen ole tarpeellista säilyttää turhaa levytilaa viemässä.

Viimeisenä valintana oleva *delete unknown attributes* -valinta puolestaan poistaa koodista kaikki muuttujat, joita KlassMaster ei tunnista standardimuuttujiksi. Lähdekoodissa sellaisia ei ole mahdollista olla. Kuitenkin jotkin ulkopuoliset tahot, kuten kääntäjä, voivat sellaisia tavukooditietoihin lisätä, vaikkei niitä koodin ajonaikaisessa suorituksessa tarvitakaan. Tästä syystä myös nämä tunnistamattomat muuttujat voitiin valita poistettaviksi tavukooditiedostoista. [19.]

6.2.2 Trimin suorittaminen

Asetusvalintojen jälkeen Trim-toimenpide ajettiin onnistuneesti läpi. Toimenpiteen tulokset eli tiedot siitä, mitkä luokat, muuttujat ja metodit kutistettiin, näkyivät KlassMasterin graafisessa käyttöliittymässä (kuva 9). Ohjelma tuotti tästä toimenpiteestä myös tekstimuotoisen logitiedoston.



Kuva 9. KlassMasterin Trim report -ikkuna, jossa tiedot Primavistan kutistamisesta

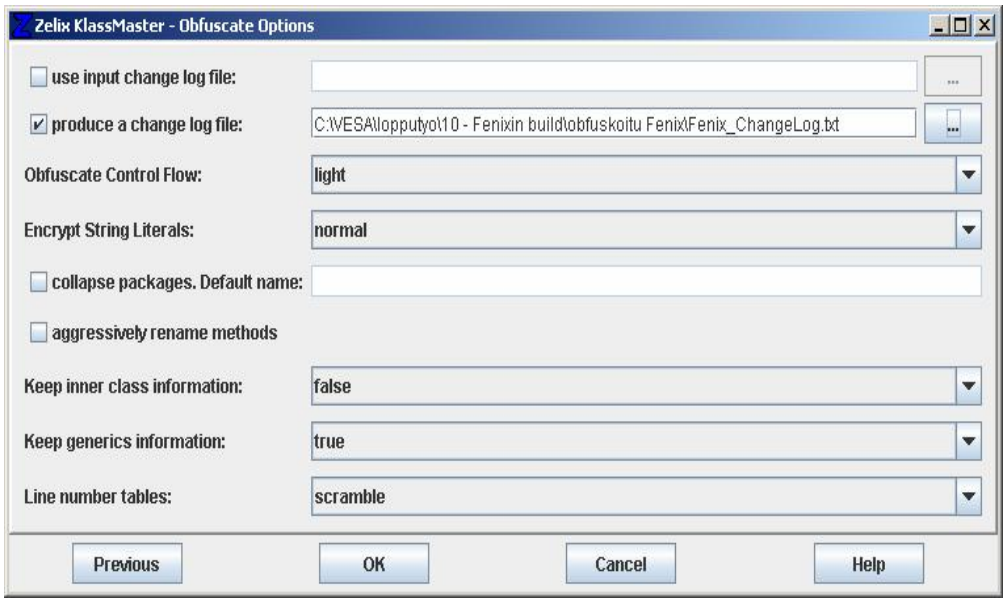
6.3 Obfuscate-toimenpide

Kun Trim-toimenpide saatiin ajettua, seuraava toimenpide oli varsinainen obfuskointi eli koodin sotkeminen. Edellä mainittujen Class.getName()-komentojen aiheuttamiin mahdollisiin ongelmiin ei vielä tässä vaiheessa kiinnitetty huomiota.

Obfuskointi ajettiin läpi obfuskoimalla kaikki Primavistan pakkaukset, luokat, metodit sekä muuttujat lukuunottamatta KlassMasterin itsensä automaattisesti poisjättämiä tapauksia sekä sovelluksen main-metodien nimiä. Tällaisia KlassMasterin automaattisesti poisjättämiä tapauksia saattavat olla esimerkiksi RMI-koodia sisältävät luokat [19]. Main-metodien nimet puolestaan jätettiin obfuskoimatta, koska niiden nimien muuttamisesta saattaisi seurata, ettei Java-virtuaalikone löytäisi näitä main-metodeja. Siten virtuaalikone ei myöskään osaisi käynnistää sovellusta. Tällainen tilanne tuli eteen obfuskointityökalujen vertailuvaiheessa, kun KlassMasteria käytettiin ensimmäistä kertaa testisovelluksen kanssa.

6.3.1 Obfuscate-asetukset

Obfuskoinnissa käytettiin seuraavia asetuksia (kuva 10):



Kuva 10. KlassMasterin Obfuscate-ikkuna valituilla asetuksilla

Valitsematta jäänyt valinta *use input change log file* tarkoittaa, että obfuskoinnin yhteydessä luettaisiin edellisen obfuskoinnin muuttuneet nimet tekstitiedostosta. Ohjelma tuottaa tällaisen tekstitiedoston aina obfuskoinnin yhteydessä. Tämän tiedoston avulla voitaisiin obfuskoinnissa aina käyttää edellisen obfuskointikerran kanssa yhtenäistä nimeämistapaa, kun obfuskointi muuttaa pakkausten, luokkien, metodien ja muuttujien nimiä.

Yhtenäisestä nimeämistavasta olisi hyötyä sellaisissa tapauksissa, joissa esimerkiksi RMI:n toimimiseksi tiettyjen luokkien nimet pitäisi muuttaa aina samalla tavalla. Esimerkki tällaisesta tilanteesta on, jos ainoastaan asiakaspuolen koodi obfuskoitaisiin, mutta asiakaspuolen kanssa RMI:n välityksellä kommunikoivan serveripuolen koodia ei obfuskoitaisi. Koska Primavistan tapauksessa kummatkin RMI:n välityksellä kommunikoivat osapuolet obfuskoidaan aina yhtä aikaa, ei tätä toimintoa ole tarpeellista käyttää.

Valittu *produce a change log file* -valinta puolestaan tuottaa tämän edellä mainitun tekstitiedoston, jossa on tiedot kaikista obfuskoinnin myötä muuttuneista nimistä. Tämä tekstitiedosto on tärkeä etenkin Stack trace -ilmoitusten kääntämiseksi viittaamaan alkuperäiseen koodiin.

Obfuscate Control Flow -valinta tarjoaa vaihtoehdoiksi *light*, *normal* ja *aggressive*. Näillä vaihtoehdoilla määritellään KlassMasterin käyttämän *Flow Obfuscation* -tekniikan tehokkuusaste eli miten voimakkaasti sitä käytetään. Vaihtoehto *aggressive* on tehokkain ja vastaavasti *light* kevyin vaihtoehto. *Flow Obfuscation* sotkee tavukoodin varsinaista rakennetta ja toimintalogiikkaa, jolloin tehokkaimmillaan käytettynä obfuskoitavan sovelluksen käyttönopeus saattaa laskea jopa 25 %:lla. Tästä syystä vaihtoehtoja *aggressive* ja *normal* ei haluttu käyttää. Valittu *light*-vaihtoehto aiheuttaa sovelluksen käyttönopeuden hidastumista ainoastaan noin yhden prosenttiyksikön verran.

String Encryption -tekniikalla merkkijonojen salaamisesta vastaava *Encrypt String Literals* -valinta tarjoaa vaihtoehdoiksi *normal*, *aggressive* ja *flow obfuscate*. Vaihtoehdoilla määritellään, miten tehokkaasti merkkijonojen sisältö halutaan enkryptata.

Encrypt String Literals -valinnan vaihtoehto *normal* ainoastaan korvaa merkkijonojen sisällöt enkryptatulla sisällöllä ja lisää tavukoodiin tiedot, miten nämä enkryptaukset puretaan suorituksen aikana. *Aggressive*-vaihtoehto puolestaan lisäksi salaa kaikki *static final String* -määreellä olevat merkkijonot, jotka saattavat jäädä salaamatta *normal*-vaihtoehdolla. KlassMasterin dokumentaatiossa kuitenkin huomautetaan, että tämä *aggressive*-vaihtoehto ei välttämättä toimi kaikilla kääntäjillä, vaikka sen toiminta onkin testattu useiden yleisimpien kääntäjien kanssa. Vaihtoehto *flow obfuscate* puolestaan toimii samoin kuin *aggressive*, mutta tässä tavukoodiin lisättävät enkryptauksen purkutiedot on sotkettu käyttämällä *Flow Obfuscation* -tekniikkaa. Tällöin jopa merkkijonojen enkryptauksen purkutietoja on hankala ulkopuolisen ymmärtää.

KlassMasterin dokumentaatiossa kuitenkin huomautetaan myös *flow obfuscate* -vaihtoehdon kohdalla ongelmista. Niitä saattaa syntyä joidenkin sovellusta ajavien Java-virtuaalikoneiden ja Javan ajonaikaisesta käännöksestä konekielelle vastaavien JIT:ien (Just-in-time compiler) kanssa. Varmuuden vuoksi *aggressive* ja *flow obfuscate* -vaihtoehtoja ei otettu käyttöön niiden aiheuttamien mahdollisten ongelmien vuoksi, vaan käytettiin *normal*-vaihtoehtoa.

Valitsematta jäänyt *collapse packages* -valinta siirtää alipakkausten luokat ylipakkausten luokiksi. Koska tällöin pakkausten nimet lyhenevät, myös tavukooditiedostojen koko pienenee jonkin verran. Tämä saattaa kuitenkin aiheuttaa ajonaikaisia ongelmia. Niitä voi ilmetä, mikäli koodissa on viittauksia obfuskoinnin jälkeen samannimisiksi muuttuneisiin luokkiin, jotka on *collapse packages* -toiminnon myötä siirretty saman ylipakkauksen alle. Tästä syystä tämä valinta jätettiin valitsematta.

Valinta *aggressively rename methods* nimeää obfuskoidessa metodeita samoilla nimillä, mikä pienentää tavukooditiedostojen kokoa. Myös tämän valinnan kanssa saattaa tulla ongelmia, mikäli koodissa on viittauksia tämän valinnan myötä samoilla nimillä nimettyihin metodeihin. Tästä syystä tätäkään toimintoa ei valittu käytettäväksi.

Valinnalla *Keep inner class information* on vaihtoehdot *true* ja *false*. *True* tarkoittaa, että kaikki tiedot Java-koodin sisäluokista säilytetään. Näitä tietoja ei kuitenkaan tarvita tavukooditasolla ollenkaan, vaan niitä käyttävät ainoastaan kääntäjät ja muut vastaavat sovellukset. Näitä sisäluokkien tietoja ei siis ollut tarpeellista säilyttää tavukooditiedostoissa obfuskoinnin jälkeen, sillä ne vievät vain turhaan ylimääräistä levytilaa. Tälle valinnalle valittiin siis vaihtoehto *false*.

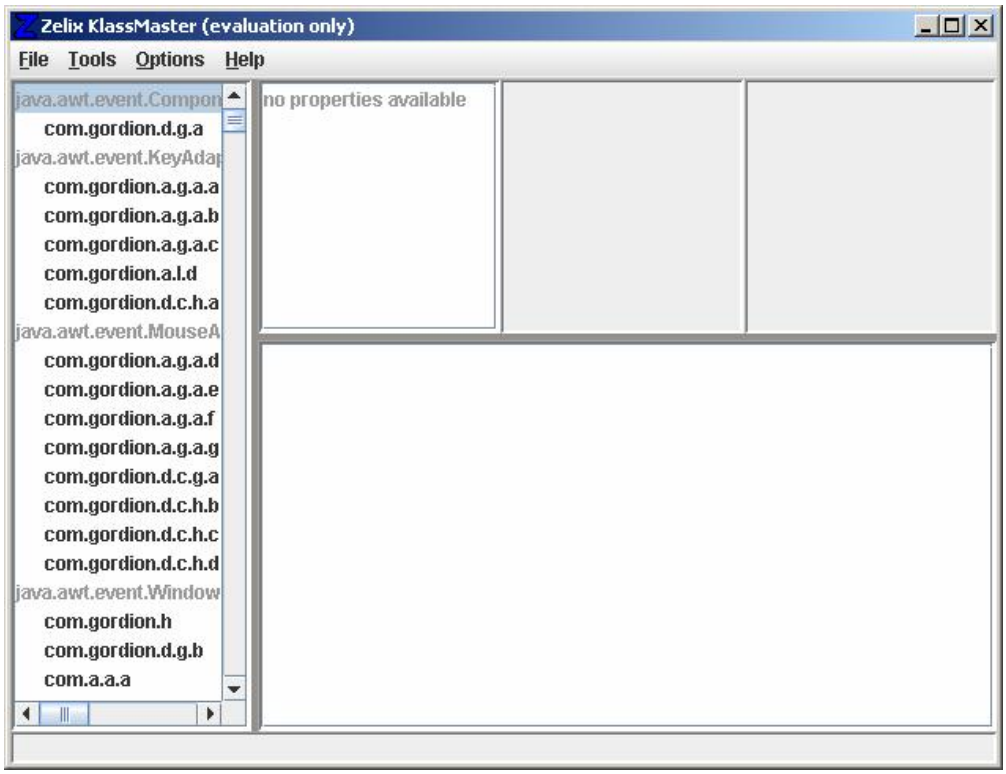
Valinta *Keep generics information* on periaatteeltaan samanlainen kuin *Keep inner class information*. Tämä valinta kuitenkin kohdistuu Javan versiosta 1.5 lähtien mukana olleisiin geneerisyystietoihin, joita sisäluokkien tietojen tapaan käyttävät ainoastaan jotkin ulkoiset sovellukset. Näiden tietojen avulla esimerkiksi kääntäjät osaavat ilmoittaa varoituksen, mikäli geneerisyyden edellyttämää tietorakenteiden tyypitystä ei ole lähdekoodissa käytetty. Jotkin J2EE (Java Enterprise Edition) -ympäristöt saattavat kuitenkin tarvita näitä geneerisyystietoja. Siksi varmuuden vuoksi niitä ei poistettu, vaan tämä valinta jätettiin *trueksi*. [19.]

Viimeisin valinta, *Line number tables* tarkoittaa tavukooditiedostoissa olevia lähdekoodiin viittaavia rivinumeroita. Näitä rivinumeroita tarvitaan Stack trace -ilmoitusten yhteydessä, jotta tiedetään, mikä lähdekoodin rivi on jonkun mahdollisen virheen aiheuttanut. Rivinumerot kuitenkin voivat tarjota takaisinkääntäjälle lisäinformaatiota takaisinkäännetyistä luokista. Siksi obfuskointityökalut yleensä poistavatkin nämä rivinumerotiedot kokonaan.

Line number tables -valinnassa oli vaihtoehdot *delete*, *keep* ja *scramble*. *Delete* poistaa kaikki nämä rivinumerotiedot ja siten pienentää tavukooditiedostojen kokoa, kun taas *keep* nimensä mukaisesti säilyttää kaikki tiedot. Valittu vaihtoehto, *scramble* puolestaan säilyttää rivinumerotiedot, mutta sotkee ne erilaisiksi ja kirjoittaa tekstitiedostoon tiedot tavasta, jolla ne on sotkettu. Tätä tekstitiedostoa tarvitaan, kun käytetään KlassMasterin Stack Trace Translation -työkalua kääntämään Stack trace -ilmoituksen viittaamaan alkuperäiseen koodiin ja sen rivinumeroihin. Täten *scramble*-vaihtoehdon avulla saadaan säilytettyä rivinumerot mahdollisia Stack trace -ilmoituksia varten. Toisaalta *scramble*-vaihtoehdolla kuitenkin sotketaan rivinumerot, ettei niistä takaisinkääntäessä ole läheskään niin suurta hyötyä kuin normaalisti. [19.]

6.3.2 Obfuskoinnin suorittaminen ja obfuskoitujen JAR-pakettien ajaminen

Asetusvalintojen jälkeen Obfuscate-toimenpide suoritettiin onnistuneesti läpi. Obfuskoinnin jälkeen KlassMasterin pääikkunassa oli nähtävillä, että luokkien nimet olivat muuttuneet erilaisiksi (kuva 11).



Kuva 11. KlassMasterin pääikkuna obfuskoinnin jälkeen

Obfuskoituja JAR-paketteja kovalevylle tallentaessa ohjelma ilmoitti varoituksen siitä, että tiedosto DatabaseClient.jar oli digitaalisesti allekirjoitettu (*signed*) ennen obfuskointia. Obfuskoinnin jälkeen tuo allekirjoitustieto pois-

tui tiedostosta ja se olisi uudelleen allekirjoitettava. Tämä varoitus johtui siitä, että asiakaspuolen koodin sisältävä DatabaseClient.jar -paketti on alun perin allekirjoitettu Primavistan build-vaiheessa Javan Web Start -tekniikkaa käyttäviä JNLP-tiedostoja varten. Kaikki tällaiset allekirjoitustiedot katoavat obfuskoinnin aikana. Tämä ongelma vältettiin, kun obfuskointi liitettiin osaksi Primavistan build-vaihetta. Tällöin obfuskoinnin tuloksena syntynyt uusi DatabaseClient.jar -tiedosto oli allekirjoitettava uudelleen obfuskoinnin jälkeen.

Obfuskoinnin tuloksena syntyivät uudet, obfuskoidut JAR-tiedostot EntryServer.jar ja DatabaseClient.jar. Kutistuksen ja obfuskoinnin jälkeen kummankin tiedoston koko oli pienentynyt noin 10 %:lla alkuperäisestä. Näiden obfuskoitujen JAR-tiedostojen ajaminen ei kuitenkaan enää onnistunut, vaan konsolille tulostui virheilmoitus:

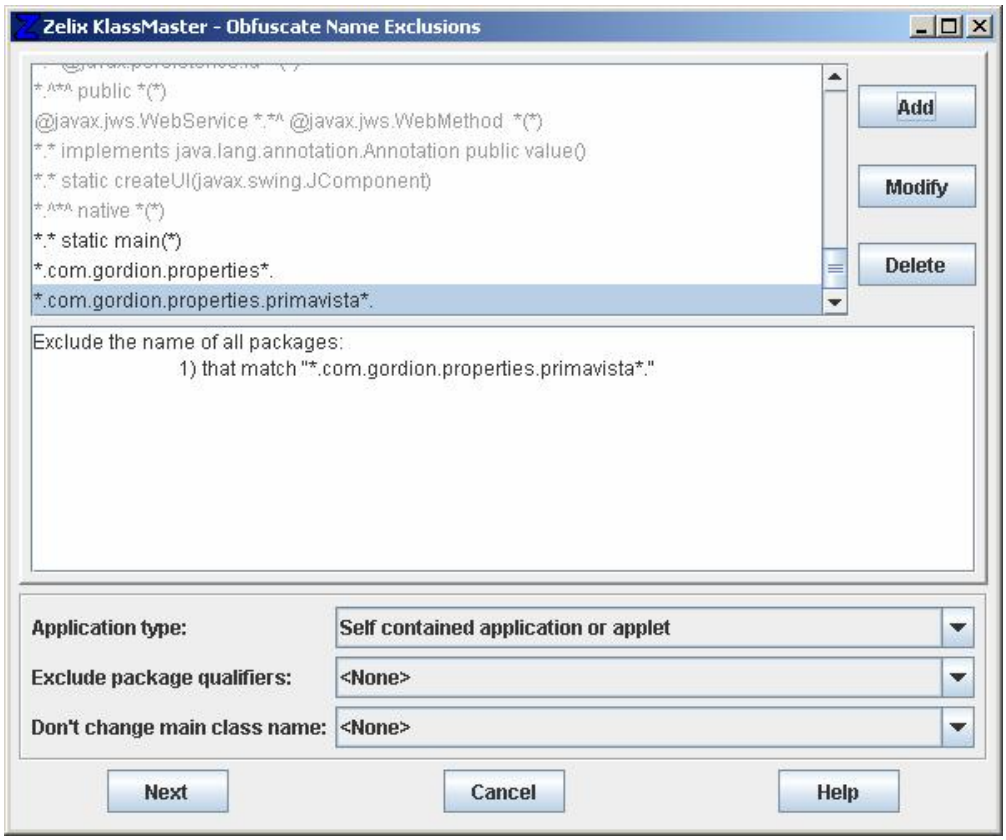
```
Exception in thread "main"
java.lang.ExceptionInInitializerError
Caused by: java.util.MissingResourceException: Can't find
com/primavista/client/MessageCatalog bundle
      at java.util.logging.Logger.setupResourceInfo(
         Logger.java:1309)
      at java.util.logging.Logger.<init>(Logger.java:204)
      at java.util.logging.Logger.getLogger(Logger.java:300)
      at com.a.b.a.<clinit>(Unknown Source)
```

Tämä virhe liittyi Javan properties- eli resurssitiedostojen käyttöön Primavistan koodissa. Näihin resurssitiedostoihin voi sisällyttää esimerkiksi koodissa käytettävien hakemistopolkujen tai merkkijonojen sisällöt [22]. Tällöin niitä ei tarvitse kirjoittaa kovakoodatusti itse koodiin, vaan ne voi aina hakea dynaamisesti resurssitiedostoista. Itse resurssitiedostoja voidaan kuitenkin käyttää koodissa ainoastaan kovakoodatusti suoraan niihin ja niiden hakemistopolkuihin viittaamalla. Tällöin niiden suhteen ilmenee ongelmia koodia obfuskoidessa. Seuraavassa on esitetty esimerkki tavasta, jolla resurssitiedostoja käsitellään Primavistan koodissa:

```
PropertyResourceBundle bundle =
(PropertyResourceBundle)ResourceBundle.getBundle(
 "com/gordion/gui/panel/usermanagement/ChangePasswordPanel");
```

Tällaisissa tapauksissa ongelmat johtuvat siitä, kun esimerkiksi pakkauksen com.gordion.gui.panel.usermanagement nimi muuttuu obfuskoinnin jälkeen täysin uuteen muotoon, kuten com.gordion.a.b.c. Tällöin haettua resurssitiedostoa ei enää löydy koodissa olevan kovakoodatun, obfuskoimattoman pakkauspolun mukaisesta paikasta. Ongelma päätettiin ratkaista siten, että kaikki Primavistan käyttämät resurssitiedostot siirrettiin kahden kokonaan

uuden pakkauksen alle. Näin ollen näiden pakkausten – com.gordion.properties ja com.gordion.properties.primavista – nimet voitiin jättää kokonaan obfuskoimatta (kuva 12). Tämän jälkeen resurssitiedostojen käytöstä aiheutuvaa ongelmaa ei enää ilmennyt. Luonnollisesti myös koodin kovakoodatut viittaukset käytettäviin resurssitiedostoihin piti muokata vastaamaan näitä uusia pakkauspolkuja.



Kuva 12. KlassMasterin Obfuscate Name Exclusions -ikkuna, jossa uudet resurssitiedostojen pakkaukset valittuina poisjätettäviksi

Kun obfuskointi oli tehty uudelleen resurssitiedostojen siirtämisen jälkeen, sovelluksen kummatkin obfuskoidut JAR-tiedostot saatiin ajettua ongelmitta. Pintapuolisesti tarkasteltuna sekä palvelinpuolen sovellus että asiakaspuolen sovellus näyttivät toimivan täysin normaalisti. Edes aiemmin mainitut Class.getName()-komennot eivät tuottaneet mitään ongelmia: esimerkiksi näiden komentojen avulla käytettävä Primavistan graafinen tilarivi toimi normaalisti. Myös RMI-koodi toimi täysin normaalisti, sillä palvelinpuolen ja asiakaspuolen sovellukset kommunikoivat toimivasti keskenään RMI:n välityksellä. Jos RMI ei olisi toiminut, asiakassovelluksella ei oltaisi nähty mitään tietokannoista haettua dataa.

Ensimmäinen Primavistan obfuskointi onnistui lähes ongelmitta, ja sen pintapuolinen toiminta ei näyttänyt muuttuvan obfuskoinnin myötä ollenkaan. Täten yritykselle voitiin päättää hankittavan Zelix KlassMasterin kaupallinen versio. Huomattavaa kuitenkin oli, että vielä lopullisen obfuskoinnin jälkeen oli tehtävä varsinainen tarkempi ja järjestelmällisempi Primavistan testaaminen.

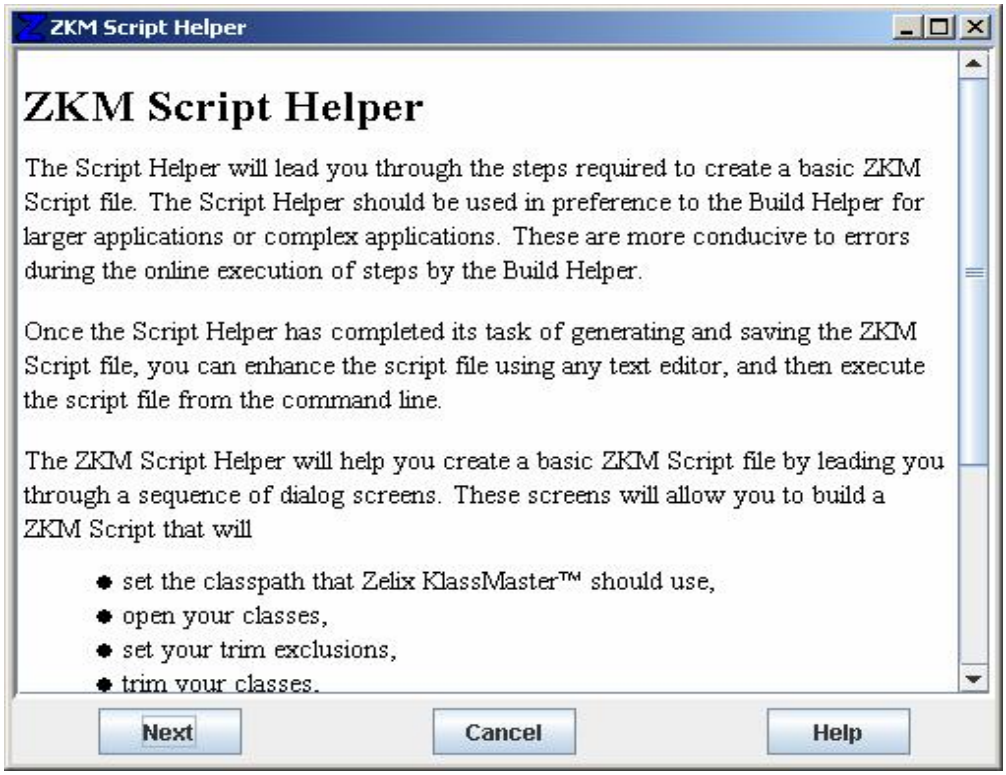
7 PRIMAVISTAN LOPULLINEN OBFUSKOINTIPROSESSI

Primavistan varsinainen lopullinen obfuskointiprosessi voitiin aloittaa, kun Gordionille oltiin hakittu Zelix KlassMasterin kokonainen kaupallinen versio. Prosessin tarkoituksena oli ensin toteuttaa obfuskointi kertaalleen yhdellä paikallisella työasemalla siten, että se liitettäisiin kiinteäksi osaksi Primavistan Ant-buildia. Jotta KlassMasterilla obfuskoiminen voitaisiin ajaa Antin kautta ei-graafisena toimenpiteenä automaattisesti, KlassMasterilla oli luotava script-tiedosto tätä varten.

Kun obfuskointi saatiin onnistuneesti ajettua Ant-buildin kautta, oli sen jälkeen testattava järjestelmällisesti obfuskoidun Primavistan toiminta. Kun testaus oli suoritettu ja Primavistan toiminta havaittu normaaliksi, viimeinen työvaihe oli siirtää kaikki obfuskoinnissa tarvittava materiaali versionhallintaan. Tällöin yrityksen muutkin kehitystiimin jäsenet pystyisivät tekemään obfuskoinnin omilta paikallisilta työasemiltaan.

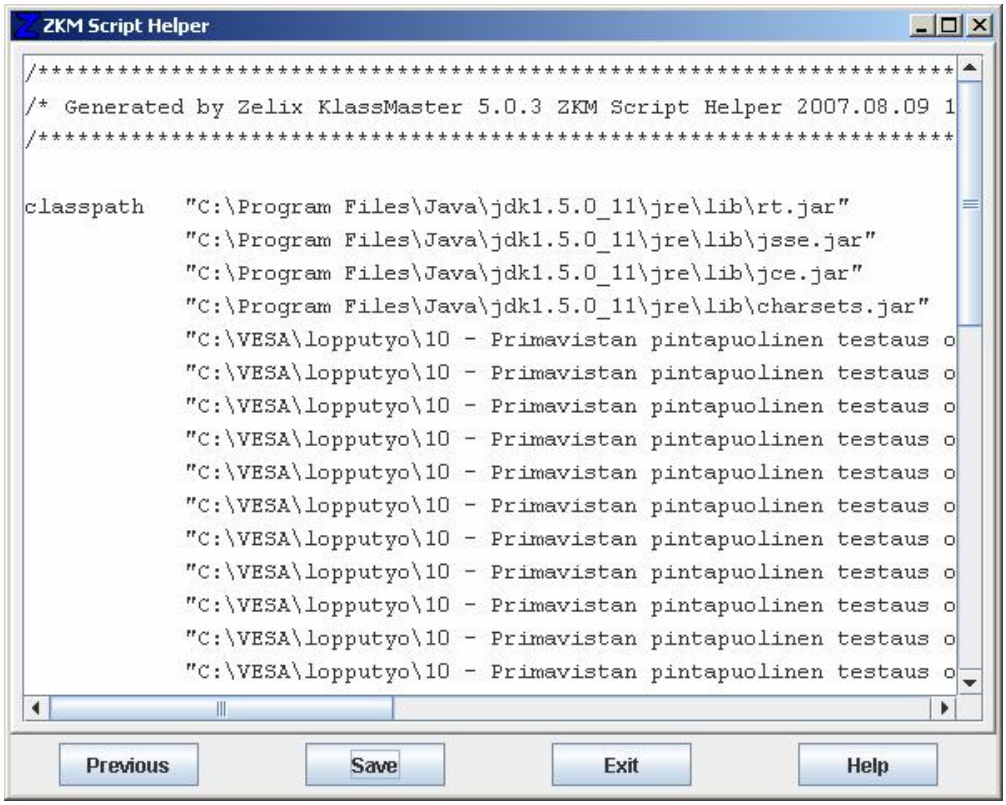
7.1 Zelix KlassMasterin ZKM-script-tiedosto

Zelix KlassMasteria voi käyttää komentorivityökaluna script-tiedosto ZKM-scriptin avulla. Tästä tekstitiedostosta käy ilmi kaikki samat asiat, jotka tehtiin edellä esitetyssä obfuskoinnissa KlassMasterin graafisella käyttöliittymällä. Tiedoston voi luoda itse kirjoittamalla käyttäen KlassMasterin määräämiä muotoilumalleja tekstille, mutta helpoin tapa tehdä tiedosto on käyttää ZKM Script Helper -työkalua. Sen graafisella käyttöliittymällä (kuva 13) voidaan ensin valita kaikki obfuskoinnissa käytettävät tiedostot ja asetukset, kuten normaalistikin KlassMasteria käyttämällä. Lopuksi nämä tiedot tallennetaan tämän script-tiedoston muotoiseksi tekstitiedostoksi.



Kuva 13. ZKM Script Helper -työkalun aloitusikkuna

Kaikki ZKM-scriptin tekemiseksi vaadittavat vaiheet olivat täysin samanlaisia kuin aiemmassakin Primavistan testiobfuskoinnissa. Ensimmäisessä vaiheessa valittin kaikki Primavistan käyttämät apukirjastot KlassMasterin classpath-ikkunassa. Tämän jälkeen avattiin obfuskoitaviksi haluttavat JAR-pakkaukset (EntryServer.jar ja DatabaseClient.jar). Seuraavaksi siirryttiin Trim-toimintoon, jonka asetuksista valittiin kaikki samat vaihtoehdot kuin aiemminkin. Samoin tehtiin viimeisen vaiheen, Obfuscate-toiminnon asetuksille. Tämän jälkeen script-tiedosto oli valmis (kuva 14) ja se voitiin tallentaa kovalevylle.



Kuva 14. ZKM Script Helperin ikkuna, jossa on valmis ZKM-script

Huomattavaa kuitenkin oli, että tähän script-tiedostoon syötettiin paikallisen työaseman hakemistopolkuja. Nämä polut kuvaavat esimerkiksi niitä hakemistoja, joista löytyy Primavistan käyttämät apukirjastot tai Primavistan JAR-pakkaukset.

Näitä yhden paikallisen työaseman hakemistopolkuja ei luonnollisestikaan voitu käyttää enää siinä vaiheessa, kun tätä script-tiedostoa käytettiin kaikkien yrityksen kehitystiimin jäsenten työasemilta. Sen vuoksi kaikki hakemistopolut piti kirjoittaa suhteelliseen muotoon sellaisiksi, että ne voidaan erikseen määritellä Ant-buildissa vastaamaan kunkin työaseman paikallisia hakemistoja. Esimerkiksi Javan kotihakemisto ei voi olla muotoa *C:\Program Files\Java*, vaan se pitää olla muodossa *%java.dir%*. Tällöin Antin avulla saadaan määriteltyä kultakin työasemalta oma tarkka Javan kotihakemisto. Tällaiseen muotoon muokattu ZKM-script-tiedosto on esitetty seuraavassa:

```
/************************************************************
****************/
/* Generated by Zelix KlassMaster 5.0.3 ZKM Script Helper
2007.08.09 11:29:09 */
/************************************************************
****************/

classpath  "%java.dir%\lib\rt.jar"
           "%java.dir%\lib\jsse.jar"
           "%java.dir%\lib\jce.jar"
           "%java.dir%\lib\charsets.jar"
           "%libraries.dir%\commons-cli-1.0.jar"
           "%libraries.dir%\SQLBaseJDBC.jar"
           "%libraries.dir%\help.jar"
           "%libraries.dir%\hsviewer.jar"
           "%libraries.dir%\iText.jar"
           "%libraries.dir%\jcommon-1.0.8.jar"
           "%libraries.dir%\jfreechart-1.0.2.jar"
           "%libraries.dir%\jfreereport-0.8.8-01.jar"
           "%libraries.dir%\jh.jar"
           "%libraries.dir%\jlfgr-1_0.jar"
           "%libraries.dir%\jsearch.jar"
           "%libraries.dir%\kunststoff.jar"
           "%libraries.dir%\libfonts-0.2.2.jar"
           "%libraries.dir%\libformula-0.1.2.jar"
           "%libraries.dir%\libloader-0.2.2.jar";

open       "%open.dir%\EntryServer\EntryServer.jar"
           "%open.dir%\DatabaseClient\DatabaseClient.jar";

trim       deleteSourceFileAttributes=false
           deleteDeprecatedAttributes=true
           deleteAnnotationAttributes=true
           deleteUnknownAttributes=true;

exclude    *.* static main(*) and
           *.com.gordion.properties*. and
           *.com.gordion.properties.primavista*.;

obfuscate  changeLogFileIn=""
           changeLoFileOut=
           "%logs.dir%\ZKM_ObfuscationChangeLog.txt"
           keepGenericsInfo=true
           obfuscateFlow=light
           encryptStringLiterals=normal
           lineNumbers=scramble;

saveAll    archiveCompression=all "%save.dir%";
```

7.2 Obfuskoinnin liittäminen Primavistan Ant-buildiin

Zelix KlassMasterin käyttöä edellä tehdyn ZKM-script-tiedoston kanssa integroitaessa Primavistan Ant-buildiin oli otettava huomoon, missä järjestyksessä kaikki toimenpiteet tulisi tehdä. Buildissa tehdään ensin käännös Javan lähdekooditiedostoista tavukooditiedostoiksi, minkä jälkeen nämä tavukooditiedostot pakataan kahteen JAR-pakettiin. Tähän vaiheeseen liitettiin näiden JAR-pakettien obfuskointi KlassMasterilla.

Lopuksi oli vielä tehtävä edellä mainittu DatabaseClient.jar -tiedoston uudelleen allekirjoittaminen, jotta tiedosto toimisi Javan Web Start -tekniikan kanssa ongelmitta. Huomattavaa oli myös se, että koska obfuskoitavat JAR-paketit kommunikoivat keskenään RMI:n välityksellä, ne kummatkin on obfuskoitava aina samanaikaisesti. Tällöin KlassMaster käyttää kummankin JAR-paketin kanssa yhteneväistä luokkien ja metodien nimeämiskäytäntöä, mikä takaa RMI:n toimimisen obfuskoinnin jälkeenkin.

Primavistan Ant-buildiin kirjoitettiin näille uusille toimenpiteille oma osio, jossa ensin on määritelty hakemisto tools.dir. Tästä hakemistosta löytyy kunkin ohjelmoijan työasemalle tallennettava Zelix KlassMasterin JAR-tiedosto. $-merkeillä ilmoitetut hakemistot ovat Antin tulkitsemia, kyseisen buildin resurssitiedostossa tarkemmin esitettyjä hakemistoja. Seuraavassa vaiheessa määritellään kaikki obfuskoinnissa tarvittavat hakemistot. Tämän jälkeen kutsutaan tietyin KlassMasterin dokumentaation mukaisin parametrein KlassMasteria tekemään obfuskointi. Esimerkiksi parametri *scriptFileName* kertoo, mistä hakemistosta löytyy käytettävä ZKM-scriptin tiedosto.

Obfuskoinnin jälkeen uudelleen allekirjoitetaan tiedosto DatabaseClient.jar samalla tavoin kuin se tehdään aiemmassakin vaiheessa Primavistan Ant-buildissa. Viimeisessä vaiheessa siirretään uudet, obfuskoidut JAR-tiedostot vanhojen, buildin aiemmassa vaiheessa luotujen obfuskoimattomien JAR-tiedostojen päälle.

Seuraavassa on esitetty tämä Ant-buildiin kirjoitettu uusi obfuskointiosio:

```
<property name="tools.dir" location="${basedir}/tools/ZKM" />
<taskdef name="ZKM" classname="ZKMTask">
   <classpath path="${tools.dir}/ZKM.jar" />
</taskdef>

<target name="obfuscate">
   <!-- 1. The obfuscation -->
   <!-- Directories to be used with the obfuscation script for
   Zelix KlassMaster: -->
   <property name="java.dir" location="${java.home}" />
   <property name="libraries.dir" location="${lib.dir}" />
   <property name="open.dir" location="${build.apps.dir}" />
   <property name="save.dir" location="${build.apps.dir}" />
   <property name="logs.dir" location=
      "${build.dir}/obfuscation_logs" />
   <!-- Attributes that correspond to KlassMaster command line
   options: -->
   <ZKM scriptFileName=
      "${tools.dir}/obfuscation_script/ZKM_script.txt"
   logFileName="${logs.dir}/ZKM_Log.txt"
   trimLogFileName="${logs.dir}/ZKM_TrimLog.txt"
```

```
    defaultExcludeFileName="${tools.dir}/defaultExclude.txt"
    defaultDirectoryName="."
    isParseOnly="false"
    isVerbose="false"
    >
    <sysproperty key="ZKM_NEW_CHANGE_LOG_ENCODING" value="UTF-
    16" />
    </ZKM>
    <!-- 2. Re-sign DatabaseClient.jar -->
    <signjar alias="gordionkeystore" storepass="gordion"
       keystore="${gordion.keystore.file}">
       <fileset file="${build.apps.dir}/DatabaseClient.jar" />
    </signjar>
    <!-- 3. Move DatabaseClient.jar and EntryServer.jar to
    directories apps\${product} -->
    <move todir="${build.apps.dir}/EntryServer">
       <fileset file="${save.dir}/EntryServer.jar"/>
    </move>
    <move todir="${build.apps.dir}/DatabaseClient">
       <fileset file="${save.dir}/DatabaseClient.jar" />
    </move>
</target>
```

Eclipsellä Ant-buildin kautta ajettuna kyseinen osio tulostaa konsolille seuraavat ilmoitukset, joista näkyvät kaikki ajetun obfuskoinnin tiedot:

```
obfuscate:
      [ZKM] [2007.08.13 13:08:57] Zelix KlassMaster 5.0.3 -
      License #14160
      [ZKM] Single - Standard
      [ZKM] Gordion Business Consulting Ltd.
      [ZKM] Kaupintie 11 A Helsinki Finland
      [ZKM] [2007.08.13 13:08:57] Preprocessing ZKM Script
      file...
      [ZKM] [2007.08.13 13:08:57] Parsing ZKM Script file...
      [ZKM] [2007.08.13 13:08:57] Setting classpath...
      [ZKM] [2007.08.13 13:08:58] Opening classes...
      [ZKM] Opened 638 classes
      [ZKM] 204 API warnings detected during open.
      [ZKM] [2007.08.13 13:09:05] Trimming...
      [ZKM] [2007.08.13 13:09:07] Setting exclusions...
      [ZKM] [2007.08.13 13:09:07] Obfuscating classes...
      [ZKM] 1 warning detected during obfuscation.
      [ZKM] [2007.08.13 13:09:12] Saving 609 classes...
      [ZKM] 1 warning detected during save.
      [ZKM] [2007.08.13 13:09:14] Terminating normally. See
      "C:\project\work\eclipse\fenix\build\obfuscation_logs\
      ZKM_Log.txt" for more detail.
      [signjar] Signing JAR:
      C:\project\work\eclipse\fenix\build\apps\DatabaseClient
      .jar
      [move] Moving 1 file to
      C:\project\work\eclipse\fenix\build\apps\EntryServer
      [move] Moving 1 file to
      C:\project\work\eclipse\fenix\build\apps\DatabaseClient
      build:
      BUILD SUCCESSFUL
      Total time: 46 seconds
```

Kuten ilmoituksesta huomataan, tämän uuden Ant-buildin osion ajo onnistui ongelmitta ja Primavista saatiin obfuskoitua sen kautta.

7.3 Obfuskoidun Primavistan testaaminen

Obfuskoidun Primavistan testaus toteutettiin järjestelmätestauksena (liite 6). Testauksessa käytettiin yrityksen valmista testaussuunnitelmaa, jonka avulla testattiin suurin osa Primavistan graafisen käyttöliittymän toiminnoista. Koska tarkoituksena oli testata vain obfuskoidun Primavistan toimintaa, ei ollut tarpeellista tehdä mahdollisimman systemaattista järjestelmätestausta testaamalla jokaista kohdetta tarkkaan. Tässä tapauksessa riittävää oli, että jokaisen testattavan kohteen voitiin todeta ainoastaan toimivan päällisin puolin.

Kaikki testitapaukset saatiin suoritettua onnistuneesti lukuunottamatta testitapausta PV:24 (PDF-tiedoston teko). PDF-tulostuksen dialogi ei auennut ollenkaan, vaan konsolille tulostui seuraava virhe:

```
java.lang.ClassNotFoundException:
com.gordion.business.report.print.PDFExport
```

Virheestä huomattiin, että sovellus yritti löytää luokkaa com.gordion.business.report.print.PDFExport ja tuotti virheen, koska ei löytänyt sitä. Ongelma johtui siitä, että obfuskoinnin jälkeen luokan nimi oli muuttunut, mutta Primavistan PDF-tulostukseen käyttämän JFreeReport-apukirjaston resurssitiedostossa viitattiin kovakoodatusti tähän luokkaan. Tätä kovakoodausta ei kuitenkaan ollut mahdollista muuttaa. Ongelma saatiin korjatuksi siten, että KlassMasterin script-tiedostoon kirjoitettiin rivit, joilla jätettiin kutistamatta luokka PDFExport ja obfuskoimatta kyseisen luokan nimi ja koko sen pakkauspolku.

Tämänkään jälkeen PDF-tulostus ei kuitenkaan toiminut. Tällä kertaa tulostusdialogi aukesi, mutta konsolile tulostui seuraava virhe:

```
java.lang.NoSuchMethodException:
com.gordion.business.report.print.PDFExport.<init>(java.awt.
Dialog)
```

Virhe syntyi tällä kertaa siitä, että sovellus ei löytänyt etsimiään PDFExport-luokan metodeita. Virheestä voitiin päätellä, että jo pelkkä PDFExport-luokan metodien obfuskointi aiheutti ongelman. Kuten edellisessäkin tapauksessa, ongelma saatiin korjatuksi muokkaamalla KlassMasterin script-tiedostoa.

Tällä kertaa siihen kirjoitettiin rivit, joilla jätettiin kaikki kyseisen PDFExport-luokan metoditkin kutistamatta ja obfuskoimatta.

Tämän jälkeen myös PDF-tulostus toimi ongelmitta. Kaikki testitapaukset siis saatiin suoritettua ja obfuskoidun Primavistan toiminta voitiin todeta normaaliksi.

7.4 Obfuskoinnin tulokset ja siirto versionhallintaan

Obfuskoinnin tuloksena syntyi kaksi obfuskoitua JAR-tiedostoa, EntryServer.jar ja DatabaseClient.jar. Kummankin JAR-tiedoston koko oli obfuskoinnin jälkeen pienentynyt noin 10 %:lla eli 200 kilotavulla. Obfuskointi onnistui eri vaiheiden jälkeen lopulta täydellisesti. Sekä asiakas- että serverisovellus toimivat obfuskoituinakin normaalisti. Takaisinkääntäjällä avattujen obfuskoitujen JAR-pakettien luokkien lähdekoodit olivat yhtä vaikeasti ymmärrettävissä kuin testikoodinkin tapauksessa.

Jotta kaikki yrityksen kehitystiimin jäsenet saisivat samat obfuskointiin tarvittavat tiedostot käyttöönsä, ne oli siirrettävä versionhallintaan, johon käytetään Subversion-ohjelmaa. Versionhallintaan siirrettiin seuraavat tiedostot:

- build.xml, joka on uusi obfuskoinnin osalta päivitetty Ant-build
- /tools/ZKM/ZKM.jar, joka on kyseiseen hakemistoon sijoitettu KlassMasterin ajettava JAR-tiedosto
- /tools/ZKM/obfuscation_script/ZKM_script.txt, joka on kyseiseen hakemistoon sijoitettu KlassMasterin script-tiedosto.

8 YHTEENVETO

Tässä työssä oli tavoitteena löytää Gordion-talousohjaus Oy:n tarpeisiin parhaiten sopiva obfuskointityökalu, jolla voitaisiin toteuttaa koko Primavistan obfuskointiprosessi. Työtä voidaan pitää onnistuneena, sillä kaikki yrityksen työlle asettamat tavoitteet saavutettiin. Obfuskointi otettiin kiinteäksi osaksi Primavistan rakennusprosessia syksystä 2007 lähtien. Tulevaisuudessa kaikki asiakkaille toimitettavat Web-Primavistan versiot tulevat olemaan obfuskoituja sovelluksia.

Parhaan obfuskointityökalun valinta oli selkeä. Obfuskointityökaluksi hankittu Zelix KlassMaster osoittautui yrityksen tarpeisiin erinomaisesti sopivaksi vaihtoehdoksi. KlassMasterilla toteutettu obfuskointi onnistui vaivattomasti

ohjelman monipuolisuuden ja helppokäyttöisyyden ansiosta. Primavistan koodin obfuskointi aiheutti jonkin verran ongelmia, etenkin sellaisten luokkien ja metodien suhteen, joiden nimiä ei voitu muuttaa obfuskoinnin yhteydessä. Ongelmat kuitenkin huomattiin tarkasti ja järjestelmällisesti tehdyn testauksen ansiosta. Kaikki ongelmat saatiin myös korjatuksi muuttamalla obfuskoinnissa käytettyjä asetuksia. Lopulta obfuskoidun Primavistan toiminta voitiin todeta täysin normaaliksi.

Obfuskoinnista, kuten mistä tahansa muustakin koodin suojausmenetelmästä puhuttaessa, nousee esille usein kaksi eri näkökantaa: Voidaan väittää, että koodin suojaaminen on ainoastaan ajan ja rahan hukkaa, koska mikään keino ei ole tarpeeksi tehokas asiansa osaavaa takaisinkääntäjää vastaan. Toisaalta voidaan olla sitä mieltä, että koodin suojaamisesta on aina hyötyä, vaikkei sillä kaikkia takaisinkääntäjiä voitaisikaan estää – riittää, että estetään kaikkien halukkaiden pääsy lähdekoodiin käsiksi.

Vaikka koodin suojaamisen tarpeellisuudesta varmasti myöhemminkin kiistellään, eri suojaustekniikat ovat kuitenkin jatkuvasti kehittyvä ja enemmän kiinnostusta herättävä aihe. Myös koodin suojaustyökalujen kehittäjien ja takaisinkääntäjien kehittäjien välillä käydään tulevaisuudessakin jatkuvaa kilpajuoksua. Toistaiseksi takaisinkääntämisen täydelliseksi estämiseksi ei ole löydetty keinoja.

Javan kaltaisten tavukoodia käyttävien ohjelmointikielten kehitys ja käyttökohteiden mahdolliset muutokset voivat osaltaan vaikuttaa siihen, tuleeko takaisinkääntämistä enää esiintymään näiden kielten yhteydessä. Kenties tällaisten ohjelmointikielten tulevaisuus onkin pikemminkin kaikille avoimen lähdekoodin kehityksessä. Myös itse Java on ollut täysin avointa lähdekoodia toukokuusta 2007 lähtien [23].

LÄHTEET

[1] Samuelson Law, Technology and Public Policy Clinic, *Frequently Asked Questions about Reverse Engineering* [verkkodokumentti, viitattu 7.11.2007]. Saatavissa: http://www.chillingeffects.org/reverse/faq.cgi

[2] Tekijänoikeuslaki 8.7.1961/404

[3] Korpela, Jukka, *Onko reverse engineering sallittua?* [verkkodokumentti, viitattu 16.9.2007]. Saatavissa: http://www.cs.tut.fi/~jkorpela/tekoik/3.6.html

[4] Sun Microsystems, *Java Technology: Brief History of Java Technology* [verkkodokumentti, viitattu 4.11.2007]. Saatavissa: http://www.java.com/en/about/

[5] Kosonen, Pekka - Peltomäki, Juha - Silander, Simo, *Java 2: Ohjelmoinnin peruskirja*. 1. painos. Porvoo: Docendo. 2005.

[6] Roubtsov, Vladimir, *Cracking Java byte-code encryption* [verkkodokumentti]. JavaWorld.com 9.5.2003 [viitattu 4.11.2007]. http://www.javaworld.com/javaworld/javaqa/2003-05/01-qa-0509-jcrypt.html

[7] Sintes, Tony, *Create a native Java executable, revisited* [verkkodokumentti]. JavaWorld.com 12.6.2000 [viitattu 5.11.2007]. Saatavissa: http://www.javaworld.com/javaworld/javaqa/2000-06/02-qa-0609-exe.html

[8] Wisegeek, *What is Encryption?* [verkkodokumentti, viitattu 5.11.2007]. Saatavissa: http://www.wisegeek.com/what-is-encryption.htm

[9] Sun Microsystems, *Java Cryptography Architecture (JCA) Reference Guide* [verkkodokumentti, viitattu 5.11.2007]. Saatavissa: http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

[10] Gupta, Sonali, *Code Obfuscation* [verkkodokumentti]. Palisade Magazine 8/2005 [viitattu 5.11.2007]. Saatavissa: http://palisade.plynt.com/issues/2005Aug/code-obfuscation/

[11] Law, Douglas, *Protecting Java Code Via Code Obfuscation* [verkkodokumentti, viitattu 5.11.2007]. Saatavissa: http://www.cs.arizona.edu/~collberg/Research/Students/DouglasLow/obfuscation.html

[12] Zelix KlassMaster [verkkodokumentti, viitattu 6.11.2007]. Zelix KlassMaster > Features > String Encryption. Saatavissa: http://www.zelix.com/klassmaster/featuresStringEncryption.html

[13] Zelix KlassMaster [verkkodokumentti, viitattu 6.11.2007]. Zelix KlassMaster > Features > Line Number Scrambling. Saatavissa: http://www.zelix.com/klassmaster/featuresLineNumberScrambling.html

[14] Zelix KlassMaster [verkkodokumentti, viitattu 6.11.2007]. Zelix KlassMaster > Features > Stack Trace Translate. Saatavissa: http://www.zelix.com/klassmaster/featuresStackTraceTranslate.html

[15] Zelix KlassMaster [verkkodokumentti, viitattu 29.10.2007]. Zelix KlassMaster > Features. Saatavissa: http://www.zelix.com/klassmaster/features.html

[16] ProGuard [verkkodokumentti, viitattu 29.10.2007]. ProGuard > FAQ. Saatavissa: http://proguard.sourceforge.net/FAQ.html

[17] yGuard [verkkodokumentti, viitattu 29.10.2007]. Saatavissa: http://www.yworks.com/en/products_yguard_about.htm

[18] DashO [verkkodokumentti, viitattu 29.10.2007]. DashO > Features. Saatavissa: http://www.preemptive.com/products/dasho/Features.html

[19] Zelix Pty Ltd, *Zelix KlassMaster Online Documentation 5.0* [verkkodokumentti, viitattu 24.10.2007]. Saatavissa: http://www.zelix.com/klassmaster/docs/index.html

[20] Sun Microsystems, *Annotations* [verkkodokumentti, viitattu 21.10.2007]. Saatavissa: http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html

[21] Sharma, Anil, *EJB 3.0 in a nutshell* [verkkodokumentti]. JavaWorld.com 9.8.2004 [viitattu 7.11.2007]. Saatavissa: http://www.javaworld.com/javaworld/jw-08-2004/jw-0809-ejb.html

[22] Sun Microsystems, *Backing a ResourceBundle with a Properties File* [verkkodokumentti, viitattu 7.11.2007]. Saatavissa: http://java.sun.com/docs/books/tutorial/i18n/resbundle/propfile.html

[23] Sun Microsystems, *Free and Open Source Java - FAQ* [verkkodokumentti, viitattu 18.11.2007]. Saatavissa: http://www.sun.com/software/opensource/java/faq.jsp

TESTISOVELLUKSEN TAVUKOODI TAKAISINKÄÄNNETTYNÄ

```
// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 25.6.2007 15:47:06
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)
// Source File Name:   Class1.java

import java.io.PrintStream;

public class Class1
{

   public Class1()
   {
   }

   public void setString(String str)
   {
      finalString = str;
   }

   protected String getString()
   {
      return finalString;
   }

   public void printString()
   {
      System.out.println((new StringBuilder(
         String.valueOf(getString()))).append("\n").toString());
   }

   public static void main(String args[])
   {
      Class2 class2 = new Class2();
   }

   private String finalString;
}


// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 25.6.2007 15:47:31
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)
// Source File Name:   Class2.java

import java.util.Random;

public class Class2
{

   public Class2()
   {
      class1 = new Class1();
      randomGenerator = new Random();
      randomInt = randomGenerator.nextInt(10);
```

```
      number = 0;
      numberStr = "Random number: ";
      for(int i = 0; i < randomInt; i++)
         number++;

      if(number < 5)
         class1.setString((new StringBuilder(
            String.valueOf(numberStr))).append(
               number).append(" < 5").toString());
      else
      if(number == 5)
         class1.setString((new StringBuilder(
            String.valueOf(numberStr))).append(
               number).append(" = 5").toString());
      else
         class1.setString((new StringBuilder(
            String.valueOf(numberStr))).append(
               number).append(" > 5").toString());
         class1.printString();
         throw new NullPointerException("Manually created exception");
   }

   private Class1 class1;
   private Random randomGenerator;
   private int randomInt;
   private int number;
   private String numberStr;
}
```

DASHO:N OBFUSKOIMA TAVUKOODI TAKAISINKÄÄNNETTYNÄ

```
// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 27.6.2007 14:01:14
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)
// Source File Name:   DashoA1*..

import java.io.PrintStream;

public class Class1
{

   public Class1()
   {
   }

   public void a(String s)
   {
      a = s;
   }

   public String a()
   {
      return a;
   }

   public void DashoEval_b()
   {
      System.out.println((new StringBuilder(
         String.valueOf(
            a()))).append(A_B_C_D("\013")).toString());
    }

   public static void main(String args[])
   {
      b b1 = new b();
   }

   public static String A_B_C_D(String s)
   {
      char ac[] = new char[s.length()];
      s.getChars(0, s.length(), ac, 0);
      char c = '\0';
      int i = 0;
      do
      {
         if(i >= ac.length)
            return new String(ac);
         ac[i] = (char)(ac[i] - 1 ^ c++);
         i++;
      } while(true);
   }

   private String a;
}
```

```
// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 27.6.2007 14:01:37
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)
// Source File Name:   DashoA1*..

import java.util.Random;

public class b
{

   public b()
   {
      a = new Class1();
      DashoEval_b = new Random();
      DashoEval_c = DashoEval_b.nextInt(10);
      DashoEval_d = 0;
      DashoEval_e = Class1.A_B_C_D("Samhli'j~eio\1778/");
      break MISSING_BLOCK_LABEL_54;
      _L4:
         a.DashoEval_b();
         throw new
            NullPointerException(
               Class1.A_B_C_D("Namwfjk\177)kyonzll1ukqrfc\177xx"));
      _L2:
         while(DashoEval_d != 5)
         {
            a.a((new StringBuilder(
               String.valueOf(DashoEval_e))).append(
                  DashoEval_d).append(Class1.A_B_C_D("!@#7")).toString());
            continue; /* Loop/switch isn't completed */
         }
         a.a((new StringBuilder(String.valueOf(
            DashoEval_e))).append(DashoEval_d).append(
               Class1.A_B_C_D("!=#7")).toString());
         continue; /* Loop/switch isn't completed */
         for(int i = 0; i < DashoEval_c; i++)
            DashoEval_d++;

         if(DashoEval_d >= 5) goto _L2; else goto _L1
      _L1:
         a.a((new StringBuilder(String.valueOf(
         DashoEval_e))).append(DashoEval_d).append(
            Class1.A_B_C_D("!>#7")).toString());
         if(true) goto _L4; else goto _L3
      _L3:
   }

   private Class1 a;
   private Random DashoEval_b;
   private int DashoEval_c;
   private int DashoEval_d;
   private String DashoEval_e;
}
```

PROGUARDIN OBFUSKOIMA TAVUKOODI TAKAISINKÄÄNNETTYNÄ

```
// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 25.6.2007 16:06:32
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)

import java.io.PrintStream;

public class Class1
{

   public Class1()
   {
   }

   public final void a(String s)
   {
      a = s;
   }

   public final void a()
   {
      System.out.println((new StringBuilder(
         String.valueOf(a))).append("\n").toString());
   }

   public static void main(String args[])
   {
      new a();
   }

   private String a;
}


// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 25.6.2007 16:07:29
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)

import java.util.Random;

public final class a
{

   public a()
   {
      a = new Class1();
      b = new Random();
      c = b.nextInt(10);
      d = 0;
      e = "Random number: ";
      for(int i = 0; i < c; i++)
         d++;

         if(d < 5)
            a.a((new StringBuilder(
```

```
                String.valueOf(e))).append(d).append(" < 5").toString());
          else
          if(d == 5)
             a.a((new StringBuilder(
                String.valueOf(e))).append(d).append(" = 5").toString());
          else
             a.a((new StringBuilder(
                String.valueOf(e))).append(d).append(" > 5").toString());
          a.a();
          throw new NullPointerException("Manually created exception");
       }

       private Class1 a;
       private Random b;
       private int c;
       private int d;
       private String e;
    }
```

YGUARDIN OBFUSKOIMA TAVUKOODI TAKAISINKÄÄNNETTYNÄ

```
// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 26.6.2007 13:18:35
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)

import java.io.PrintStream;

public class Class1
{

   public Class1()
   {
   }

   public void A(String s)
   {
      A = s;
   }

   protected String A()
   {
      return A;
   }

   public void B()
   {
      System.out.println((new StringBuilder(
         String.valueOf(A()))).append("\n").toString());
   }

   public static void main(String args[])
   {
      A a = new A();
   }

   private String A;
}


// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 26.6.2007 13:24:55
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)

import java.util.Random;

public class A
{

   public A()
   {
      A = new Class1();
      D = new Random();
      C = D.nextInt(10);
      B = 0;
      E = "Random number: ";
```

```
      for(int i = 0; i < C; i++)
         B++;

      if(B < 5)
         A.A((new StringBuilder(
            String.valueOf(E))).append(B).append(
               " < 5").toString());
      else
      if(B == 5)
         A.A((new StringBuilder(
            String.valueOf(E))).append(B).append(
               " = 5").toString());
      else
         A.A((new StringBuilder(
            String.valueOf(E))).append(B).append(
               " > 5").toString());
      A.B();
      throw new NullPointerException("Manually created exception");
   }

   private Class1 A;
   private Random D;
   private int C;
   private int B;
   private String E;
}
```

ZELIX KLASSMASTERIN OBFUSKOIMA TAVUKOODI TAKAISINKÄÄNNETTYNÄ

```
// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 27.6.2007 11:04:33
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)
// Source File Name:   a.java

import java.io.PrintStream;

public class a
{

   public a()
   {
   }

   public void a(String s)
   {
      a = s;
   }

   protected String a()
   {
      return a;
   }

   public void b()
   {
      int i = b;
      System.out.println((new StringBuilder(
         String.valueOf(a()))).append("\n").toString());
      if(i != 0)
         b.f = !b.f;
   }

   public static void main(String args[])
   {
      b b1 = new b();
   }

   private String a;
   public static int b;
}


// Decompiled by DJ v3.7.7.81 Copyright 2004 Atanas Neshkov  Date:
// 27.6.2007 11:04:50
// Home Page : http://members.fortunecity.com/neshkov/dj.html  - Check
// often for new version!
// Decompiler options: packimports(3)
// Source File Name:   b.java

import java.util.Random;

public class b
{

   public b()
```

```
{
   int i;
   int j;
   j = a.b;
   super();
   a = new a();
   b = new Random();
   c = b.nextInt(10);
   d = 0;
   e = z[4];
   i = 0;
   if(j == 0) goto _L2; else goto _L1
   _L1:
      d++;
      if(true)
      continue; /* Loop/switch isn't completed */
      d;
      1;
      JVM INSTR iadd ;
      d;
      i++;
   _L2:
      if(i < c) goto _L1; else goto _L3
   _L3:
      this;
      if(j != 0) goto _L5; else goto _L4
   _L5:
      this;
   _L4:
      d;
      5;
      JVM INSTR icmpge 139;
      goto _L6 _L7
   _L6:
      break MISSING_BLOCK_LABEL_96;
   _L7:
      break MISSING_BLOCK_LABEL_139;
      a.a((new StringBuilder(
         String.valueOf(e))).append(d).append(z[3]).toString());
      if(j == 0)
         break MISSING_BLOCK_LABEL_229;
      if(d == 5)
      {
         a.a((new StringBuilder(
            String.valueOf(e))).append(d).append(z[0]).toString());
         if(j == 0)
            break MISSING_BLOCK_LABEL_229;
      }
      a.a((new StringBuilder(
         String.valueOf(e))).append(d).append(z[2]).toString());
      a.b();
      throw new NullPointerException(z[1]);
}

private a a;
private Random b;
private int c;
private int d;
private String e;
public static boolean f;
private static final String z[];
```

```
static
{
   String as[];
   as = new String[5];
   as[0] = "\17752=";
   as[1] = "\022i|}\n3dk(\b-ms|\016;(wp\b:xfa\0041";
   as[2] = "\17762=";
   as[3] = "\17742=";
   as[4] = "\ri|l\0042(|}\006=m`2K";
   z = as;
   break MISSING_BLOCK_LABEL_163;
   local;
   toCharArray();
   JVM INSTR dup ;
   JVM INSTR arraylength .length;
   JVM INSTR swap ;
   int i = 0;
   JVM INSTR swap ;
   JVM INSTR dup_x1 ;
   1;
   JVM INSTR icmpgt 142;
   goto _L1 _L2
   _L1:
      JVM INSTR dup ;
      i;
   _L4:
      JVM INSTR dup2 ;
      JVM INSTR caload ;
      byte byte0;
      switch(i % 5)
      {
         case 0: // '\0'
            byte0 = 0x5f;
            break;

         case 1: // '\001'
            byte0 = 8;
            break;

         case 2: // '\002'
            byte0 = 18;
            break;

         case 3: // '\003'
            byte0 = 8;
            break;

         default:
            byte0 = 107;
            break;
      }
      byte0;
      JVM INSTR ixor ;
      (char);
      JVM INSTR castore ;
      i++;
      JVM INSTR swap ;
      JVM INSTR dup_x1 ;
      JVM INSTR ifne 142;
      goto _L3 _L
```

```
  _L3:
     JVM INSTR dup2 ;
     JVM INSTR swap ;
     goto _L4
  _L2:
     JVM INSTR swap ;
     JVM INSTR dup_x1 ;
     i;
     JVM INSTR icmpgt 65;
     goto _L5 _L1
  _L5:
     JVM INSTR new #43  <Class String>;
     JVM INSTR dup_x1 ;
     JVM INSTR swap ;
     String();
     intern();
     JVM INSTR swap ;
     JVM INSTR pop ;
     JVM INSTR ret 0;
}
```

OBFUSKOIDUN PRIMAVISTAN TESTAUSRAPORTTI

ID	TESTATTAVA KOHDE	ODOTETTU TULOS	HAVAITTU TULOS
PV:1	Lisenssitiedosto	Ohjelma päästää kirjautumaan sisään eri käyttäjätunnuksilla kyseisten tunnusten lisenssi-oikeuksilla	OK
	Pääikkunan toiminnot		
PV:2	Vastuualuelista	Kaikki listan valinnat toimivat	OK
PV:3	Vastuualuelistan täydentäminen	Listan täydentäminen onnistuu	OK
PV:4	Raporttilista	Kaikki listan valinnat toimivat	OK
PV:5	Raporttilistan täydentäminen	Listan täydentäminen onnistuu	OK
PV:6	Raporttityypin huomioiminen	Kaikki raporttityypit toimivat	OK
PV:7	Sarakemallilista	Kaikki listan valinnat toimivat	OK
PV:8	Sarakemallin lukitus	Lukitus onnistuu	OK
PV:9	Tilarivi	Tilarivin huomautukset ja virheilmoitukset toimivat	OK
PV:10	Raportoitavan kauden valinta	Kauden valinta onnistuu	OK
PV:11	Raportoitavan tiedon valinta - sarakemallit	Raporttien data vaihtuu sarakevalinnan mukaan	OK
PV:12	Muut pääikkunan toiminnot	Kaikki pääikkunan toiminnot toimivat	OK
PV:13	Pääikkunan toimintopainikkeet	Kaikki pääikkunan painikkeet toimivat	OK
	Raporttitoiminto		
PV:14	Piilota rivit/sarakkeet -asetus	Rivien/sarakkeiden piilottaminen onnistuu	OK
PV:15	Zoomaus vastuualueiden mukaan	Zoomaus onnistuu	OK
PV:16	Monen rivin zoomaus	Zoomaus onnistuu	OK
PV:17	Zoomaus rivin mukaan	Zoomaus onnistuu	OK
	Sarakemallit		
PV:18	Sarakkeiden värit	Sarakkeiden värit näkyvät	OK
PV:19	Lukujen skaalaus	Lukujen skaalaus onnistuu	OK

PV:20	Sarakkeisiin tallennetut oletukset	Sarakkeiden tallennetut oletusarvot toimivat käytössä	OK
	Tulostus		
PV:21	Esikatselu	Esikatselu onnistuu	OK
PV:22	Esikatselu eri raporttityypeillä	Esikatselu onnistuu	OK
PV:23	Esikatselun asetukset	Asetukset toimivat	OK
PV:24	PDF-tiedoston teko	PDF-tiedosto saadaan tehtyä ja tallennettua kovalevylle	Virhe
	Etsintä		
PV:25	Etsintätoiminto eri listoihin	Etsintä toimii valintalistoissa	OK
PV:26	Etsintätoiminto raporttiin	Etsintä toimii raporttinäkymässä	OK
PV:27	Numeroiden etsintä	Numeroiden etsintä toimii sekä valintalistoissa että raporttinäkymässä	OK
	Raporttityypit		
PV:28	Vastuualueraportti	Raportti latautuu	OK
PV:29	Viikonpäivät näkyviin	Viikonpäivät näkyvät raporteissa	OK
PV:30	Yhteenvetoraportti	Raportti latautuu	OK
PV:31	Matriisiraportti 1	Raportti latautuu	OK
PV:32	Matriisiraportti 2	Raportti latautuu	OK
PV:33	Sarjalistausraportti 1	Raportti latautuu	OK
PV:34	Sarjalistausraportti 2	Raportti latautuu	OK
	Raporttilomakkeen toiminnot		
PV:35	Työkalujen aktivoituminen	Työkalut aktivoituvat raportin latauduttua	OK
PV:3	Lukujen syöttö	Lukujen syöttö onnistuu raportteihin	OK
PV:37	Zoomaukseen syöttö	Lukujen syöttö onnistuu zoomatuille riveille	OK
PV:38	Tietojen tallennus	Muokattujen tietojen tallennus onnistuu	OK
PV:39	Laskenta	Laskenta toimii	OK
PV:40	Vastuualueiden selaus	Vastuualueiden selaus onnistuu	OK
PV:41	Raporttilomakkeen selaus	Lomakkeen selaus onnistuu	OK
PV:42	Kommentointi	Kommentointi toimii	OK

PV:43	Kommenttien tulostaminen	Kommenttien tulostaminen onnistuu	OK
PV:44	Jakoperustesuunnittelu	Jakoperustesuunnittelu toimii	OK
PV:45	Leikepöytäkopiointi	Leikepöydälle kopiointi onnistuu	OK
PV:46	Tiedoston tallennus CSV-muodossa	Tiedoston tallennus .cvs:ksi onnistuu	OK
PV:47	Tiedoston tallennus HTML-muodossa	Tiedoston tallennus .html:ksi onnistuu	OK
PV:48	Kahden tason konsolidointi	Kahden tason konsolidointi toimii	OK
PV:49	Ruutuvaluutanmuunto	Valuutanmuunto ruudulla toimii	OK
	Sekalaiset toiminnot		
PV:50	Salasanan vaihto	Salasanan vaihto onnistuu	OK
PV:51	Help-toiminto	Help-toiminto toimii	OK
PV:52	Ohjelmistosta-toiminto	Ohjelmistosta-toiminto toimii	OK
	Grafiikka		
PV:53	Eri graafityypit	Kaikkien graafityyppien piirto toimii	OK
PV:54	Tulostus	Grafiikan tulostus toimii	OK